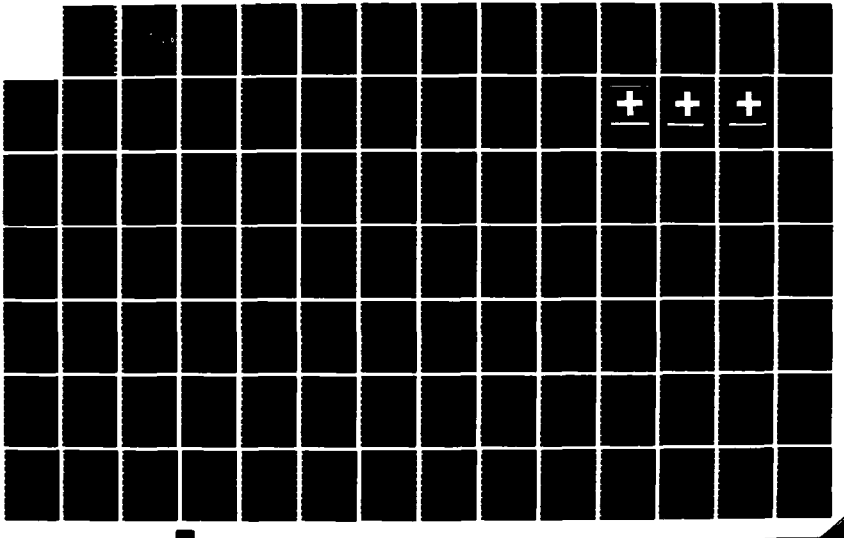
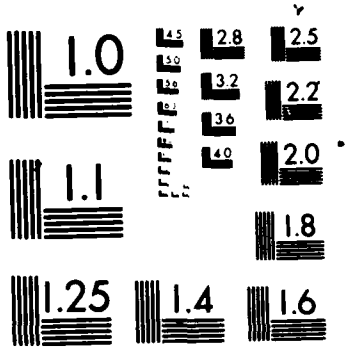
AD-A176 053 THE APPLICATION OF BIT SLICE DESIGN TO DIGITAL IMAGE PROCESSING(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M B STEWART SEP 86 1/2

UNCLASSIFIED F/G 9/2 NL






MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California




# THESIS

THE APPLICATION OF BIT SLICE DESIGN
TO DIGITAL IMAGE PROCESSING

by

Morris Bennett Stewart II

September 1986

Thesis Advisor: C.H. Lee

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION / AVAILABILITY OF REPORT<br>Approved for public release; distribution is unlimited. |
| 2b DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Postgraduate School<br>6b OFFICE SYMBOL (*If applicable*)<br>62 | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| 6c ADDRESS (*City, State, and ZIP Code*)<br>Monterey, California 93943-5000 | 7b. ADDRESS (*City, State, and ZIP Code*)<br>Monterey, California 93943-5000 |
| 8a NAME OF FUNDING / SPONSORING ORGANIZATION<br>8b OFFICE SYMBOL (*If applicable*) | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c ADDRESS (*City, State, and ZIP Code*) | 10 SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO |

11 TITLE (*Include Security Classification*)
THE APPLICATION OF BIT SLICE DESIGN TO DIGITAL IMAGE PROCESSING

12 PERSONAL AUTHOR(S)
Stewart, Morris B.

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (*Year, Month, Day*) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM _____ TO _____ | 1986 September | 129 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Am29203 Bit Slice Evaluation Board, Digital Image Processing, Execution Speed, Microroutine |
| | | | |
| | | | |

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The digital image processing requirements of today's industry are increasing at an astounding rate. With the faster satellite data transmission rates and more frequent data collection periods, both spatial storage and processing speed problems are becoming more prevalent. Digital image processing algorithms must be precise and efficient to meet these needs. This research project studies the implementation of an image smoothing algorithm as a combination of custom tailored hardware and firmware, i.e., using bit slice design.

Bit slice microprocessor design involves the configuration of very fast bit slice devices and the microprogramming necessary to command the hardware to perform a specific task. The result is a high-speed processor, but the price paid is the long and complex design time. Fixed instruction set microprocessor based design is more common but does not permit the same flexibility in hardware configuration

| 20 DISTRIBUTION / AVAILABILITY OF ABSTRACT | 21 ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL<br>Prof. C.H. Lee | 22b TELEPHONE (*Include Area Code*)<br>(408) 646-2190 / 22c OFFICE SYMBOL<br>62Le |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. or software coding. Hence, the design time is much shorter and less difficult.

The image smoothing algorithm was implemented using both bit slice and microprocessor based design. The bit slice design was performed on Advanced Micro Device's Am29203 Bit Slice Evaluation Board. The board is a 16 bit bit slice microprocessor that allows the user to create and evaluate bit slice microcode. The microprocessor based design was done on a Z-80 based microcomputer.

The bit slice design yielded a much faster system than that of the Z-80 design. The design time for the bit slice system was also much longer and much more complex than that for the Z-80 design. When making a decision as to which type of design to pursue, the dominating factor is usually the cost of the design, namely, the time and difficulty involved. In the case of digital image processing, however, the algorithms are used many times over and on huge data sets. Therefore, the extra time spent and the complexity involved in bit slice microprocessor design would be rewarded in the form of great savings in execution time when the system is put to use.

Approved for public release; distribution is unlimited.

# The Application of Bit Slice Design to Digital Image Processing

by

Morris Bennett Stewart II
Lieutenant(jg), United States Coast Guard
B.S.E.E., U. S. Coast Guard Academy, May 1982

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

Author: _____
Morris Bennett Stewart II

Approved by: _____
Chin-Hwa Lee, Thesis Advisor

_____
Mitchell L. Cotton, Second Reader

_____
Harriett B. Rigas
Chairman, Department of Electrical and Computer Engineering

_____
John N. Dyer
Dean of Science and Engineering

ABSTRACT

The digital image processing requirements of today's industry are increasing at an astounding rate. With the faster satellite data transmission rates and more frequent data collection periods, both spatial storage and processing speed problems are becoming more prevalent. Digital image processing algorithms must be precise and efficient to meet these needs. This research project studies the implementation of an image smoothing algorithm as a combination of custom tailored hardware and firmware, i.e., using bit slice design.

Bit slice microprocessor design involves the configuration of very fast bit slice devices and the microprogramming necessary to command the hardware tc perform a specific task. The result is a high-speed processor, but the price paid is the long and complex design time. Fixed instruction set microprocessor based design is more common but does not permit the same flexibilty in hardware configuration or software coding. Hence, the design time is much shorter and less difficult.

The image smoothing algorithm was implemented using both bit slice and microprocessor based design. The bit slice design was performed on Advanced Micro Device's Am29203 Bit Slice Evaluation Board. The board is a 16 bit

bit slice microprocessor that allows the user to create and evaluate bit slice microcode. The microprocessor based design was done on a Z-80 based microcomputer.

The bit slice design yielded a much faster system than that of the Z-80 design. The design time for the bit slice system was also much longer and much more complex than that for the Z-80 design When making a decision as to which type of design to pursue, the dominating factor is usually the cost of the design, namely, the time and difficulty involved. In the case of digital image processing, however, the algorithms are used many times over and on huge data sets. Therefore, the extra time spent and the complexity involved in bit slice microprocessor design would be rewarded in the form of great savings in execution time when the system is put to use.

TABLE OF CONTENTS


I. INTRODUCTION. . . . . 11

A. GENERAL BACKGROUND. . . . . 11

B. APPROACHES. . . . . 14

C. BIT SLICE DESIGN CHARACTERISTICS. . . . . 16

D. DESCRIPTION OF THE BIT SLICE EVALUATION BOARD 19

E. INTRODUCTION TO IMAGE SMOOTHING . . . . . 20

II. FUNCTIONAL DESCRIPTION OF THE BIT SLICE EVALUATION BOARD. . . . . 25

A. ARCHITECTURE. . . . . 25

1. The Computer Control Unit . . . . . 28

2. The Arithmetic Logic Unit . . . . . 29

3. The Macro Memory and Input/Output . . . . 30

B. MICROPROGRAMMING THE AM29203 EVALUATION BOARD 31

1. Example 1 . . . . . 31

2. Example 2 . . . . . 33

3. Example 3 . . . . . 35

C. SUMMARY . . . . . 36

III. BIT SLICE DEVELOPMENT OF THE IMAGE PROCESSING ALGORITHM . . . . . 38

A. THE ALGORITHM . . . . . 38

B. THE MICROROUTINE. . . . . 41

C. EXECUTION TIME. . . . . 43

D. THE 512 X 512 ARRAY . . . . . 51

IV. MICROPROCESSOR DEVELOPMENT OF THE IMAGE PROCESSING ALGORITHM . . . . . 56

A. THE CODE AND EXECUTION TIME . . . . . 56

V. CONCLUSIONS AND RECOMMENDATIONS . . . . . 60

A. EVALUATION BOARD SHORTCOMINGS . . . . . 60

B. DISCUSSION OF BIT SLICE DESIGN. . . . . 62

APPENDIX A: FORTRAN IMAGE SMOOTHING PROGRAM. . . . . 65

APPENDIX B: MICROCODE DOCUMENTATION FOR EXAMPLE 1. . . 69

APPENDIX C: DOCUMENTATION FOR THE 5 X 5 MICROROUTINE . 77

LIST OF REFERENCES. . . . . .126

BIBLIOGRAPHY. . . . . .127

INITIAL DISTRIBUTION LIST . . . . . .128

LIST OF TABLES

I. APPROXIMATE DMA LIBRARY HOLDINGS. . . . . 12

II. REGISTERS AVAILABLE . . . . . 30

III. PERMISSIBLE CLOCK PERIODS . . . . . 45

IV. INSTRUCTION TIMING ANALYSIS . . . . . 50

LIST OF FIGURES


1. Neighborhood Averaging . . . . . 21
2. The Original Uncontaminated Image. . . . . 22
3. The Original Contaminated Image. . . . . 23
4. The Contaminated Image After Smoothing . . . . . 24
5. Evaluation Board Architecture. . . . . 26
6. Three Functional Areas . . . . . 27
7. Microinstruction Format. . . . . 32
8. Microcode for Example 1. . . . . 33
9. Am2910 Instruction Set . . . . . 34
10. Microword and Documentation for Example 2. . . . 35
11. Microword and Documentation for Example 3. . . . 37
12. Memory Organization for the 5 X 5 Array. . . . . 40
13. The 5 X 5 Microroutine . . . . . 42
14. Time Path Illustration with RAM as WCS . . . . . 46
15. Time Path Illustration after PROM Substitution for WCS RAM. . . . . 48
16. Time Path Illustration with Data RAM Access. . . 49
17. Memory Organization for the 512 X 512 Array. . . 53
18. The 512 X 512 Microroutine . . . . . 54
19. The Z-80 Assembly Language Routine . . . . . 57
20. Comparison of Execution Times. . . . . 63

## ACKNOWLEDGEMENTS

I wish to gratefully acknowledge my thesis advisor, Professor Chin-Hwa Lee, who provided invaluable assistance in the completion of this thesis.

I would also like to express my gratitude to Professor Mitchell L. Cotton for his time and assistance.

And further, much thanks to Alex Cannara of Advanced Micro Devices, Inc. for his suggestions and assistance in finding answers to some difficult questions.

I would finally like to thank the Defense Mapping Agency for the symposium they held in September, 1985. The conference both conveyed the magnitude of the problems at hand as well as the state of the research presently being pursued in response to those problems. It was this conference that inspired my interest in the area of digital image processing. Such an opportunity to gather and discuss the state of the art is rare for a student such as I and was greatly appreciated.

# I. INTRODUCTION

## A. GENERAL BACKGROUND

Although the application of digital image processing techniques can be found as far back as the 1920's, it wasn't until the 1960's and the advent of the third-generation digital computer that this field received large scale interest. The digital computer offered both the speed and the storage facilities necessary to implement digital image processing algorithms on digital image data effectively. As the hardware technologies became more sophisticated, so did the many varied applications. Today, digital image processing techniques are being utilized in a wide range of professional arenas such as medical imaging, astronomy, astrophysics, cartography, as well as a myriad of military applications. The impact of this technology on industry has been received with strong enthusiasm, and new and more complex applications are being contrived every day. But some key concerns exist that need be overcome before such applications can become more than a vision, namely, the need for greater computational speed and mass storage facilities.

In the field of cartography, the Defense Mapping Agency has applied digital techniques with great success. The Defense Mapping Agency is responsible for all

mapping, charting and geodesy resources and development for the Department of Defense components as well as many other governmental agencies. In fiscal 1985, the Defense Mapping Agency was scheduled to print more than 54 million copies of thousands of maps and charts, digitize 4.4 million square nautical miles of the Earth's surface, develop 11,000 strategic points and register more than 38,000 gravity measurements. Such an undertaking resulted in approximately 7 trillion pieces of data [Ref. 1]. Table I further depicts the enormous scale of the mass storage problem at hand.

TABLE I
APPROXIMATE DMA LIBRARY HOLDINGS
[Ref. 2]

| | Inventory | New Acquisitions Per Year |
|---|---|---|
| Maps | 1,000,000 | 50,000 |
| Charts | 50,000 | 15,000 |
| Books, Periodicals | 150,000 | 40,000 |
| Geodetic Data | | |
| Control Points | 16,000,000 | 22,500 |
| Control Photos | 400,000 | 12,000 |
| Index Cards | 90,000 | 1,500 |
| Bathymetric Data | 21,000 | 400 |
| Geographic Names | | |
| On Index Cards | 4,500,000 | 150,000 |
| On Magnetic Tape | 500,000 | |
| Imagery-Cans | 100,000 | 1,700 |
| Digital Data | | |
| DTED Cells | 71,000 | 1,500 |
| DFAD Cells | 8,900 | 1,600 |
| VDD Cells | 200 | 85 |

These resources, in digitized form, are used in a number of systems such as the cruise missile guidance system, the Navy's navigational systems, and aircraft simulation systems. New uses for such digitized data are being developed, but no easy answers are readily available to the digital data collection, storage and processing problems.

To this end, the Defense Mapping Agency has launched an extensive development program to create a truly automated mapping and charting system. The purpose of such a system is to streamline the production process which is presently a labor intensive procedure. The system must incorporate both large scale data base management as well as improved methods and equipment used in the automated feature analysis. Thus, the major thrust of the project is to address the computational speed and the mass storage problems previously mentioned.

To date, the Defense Mapping Agency has been able to maintain their production schedule, but only due to the limited automation already in place. Future need for both conventional and digitized products is predicted to be increasingly heavy. To meet that need, production time must be reduced to near real time, i.e., analysis must be done at the time that the data is recorded. This will require high speed applications of very efficient digital image processing algorithms.

Presently, for example, approximately 100 million operations are required in order to run an edge detection algorithm on a 1000 X 1000 pixel image. The generation of a symbolic description of this image may require as many as 100 billion such operations. [Ref. 3] Clearly, when the amount of data on hand to be processed is considered, the processing time becomes of paramount importance. The enhancement of computational speed will be the issue that this thesis will address.

## B. APROACHES

Many hardware and software endeavors have been undertaken toward the enhancement of the computational speed dilemma associated with the processing of the large volumes of data in a digital image. Such processing, which includes image enhancement, restoration and recognition, incorporates many sets of computationally complex algorithms that consume long CPU times when executed on these large data arrays. Software applications have included very efficient machine language programming as well as advanced database management techniques. Hardware ventures have resulted in non-Von Neumann-type architectures.

Such architectural approaches toward the reduction in the time necessary to process a digital image have led to the development of special-purpose computers. Two

examples of these special-purpose computers are the array processors and the image processors. Array processors, when selected for use in digital image processing, are fast general-purpose coprocessors that are coupled to a host computer and perform the computationally intensive routines associated with image processing. They enhance the performance of the host in numerical computing tasks and achieve this high performance through parallelism and/or pipelining. Image processors, on the other hand, are more narrowly defined and are for the purpose of executing image processing routines only. [Ref. 4] An example of such an image processor is the "raster-engine" which is optimized to operate on raster-based graphics data sets.

A third architectural alternative would be the use of a supercomputer. Certainly, the execution of such digital image processing algorithms on a supercomputer would drastically reduce the required processing time. For instance, Floating Point Systems, Inc. has engineered a massively parallel supercomputer which boasts 262 billion floating-point operations per second (flops) which is a hundredfold increase over the Cray 2 recently marketed by Cray Research, Inc.[Ref. 5] It is the cost of such a system which negates this alternative in most digital

image processing applications. The Cray 2 retails for $17.6 million. Generally, most budgets in research would balk at this price tag.

As VHSIC technology approaches the fundamental limits on how small integrated circuit features can be, the research effort shifts to the study of architectural enhancements. Parallel and concurrent approaches are underway and many results from this research are already being utilized in the fields of image and signal processing.

This thesis will address the issues of implementing a specific fixed algorithm in the form of a combination of hardware and firmware for the purpose of high-speed processing. That is, the image will enter the input to this "black-box" and the processed image will exit, hopefully, in a period shorter than that achieved by a software algorithm alone. This study will utilize bit-slice hardware as the internals of this "black-box".

## C. BIT-SLICE DESIGN CHARACTERISTICS

In the design of a system for the purpose of performing digital operations, the designer has three basic building blocks from which to select. They are (1) SSI/MSI logic; (2) 8 bit, 16 bit or 32 bit fixed

instruction set microprocessors; or (3) microprogrammable bit-slice devices. Many advantages and disadvantages exist for each choice.

Should the designer choose to use SSI/MSI hardwired logic, he will be able to design any architecture imaginable having any word length he chooses and a custom tailored instruction set. The design may also have very short machine cycles on the order of 100-200 ns. The disadvantages of an SSI/MSI approach are that such a design will consume much space, be very expensive, have a long design time and be very difficult to debug. Should the designer choose to use a fixed instruction set microprocessor, the design time would be much shorter, the cost much lower, would consume much less space and would be much easier to debug. The price for these advantages is having a fixed instruction set, limited clock cycles and word lengths of only 4, 8, 16 or 32 bits.

Clearly, the advantages and disadvantages of an SSI/MSI approach are opposite to those of traditional microprocessor design. Thus, a compromise between the two is in order and it is a bit-slice approach.

Bit-slice devices are generally used in applications which require long words, special instruction sets and high speed operations. One such application is image processing. As previously detailed, high speed operations are highly desirable in image processing. Special

instruction sets would aid in the programming of image processing algorithms thus increasing the operational speed even more. Bit-slice devices permit microprogramming which allows the firmware to be custom tailored to the architecture thereby getting the most done for each clock cycle, i.e., keeping each resource busy at all times. Bit slice devices can be configured to any word length in multiples of 4, and SSI/MSI can be used to patch in any extra bit paths as needed. This gives the designer the ability to configure the architecture to any word size which translates to any pixel gray-scale range. Therefore, bit-slice devices appear to be a good candidate for the internals of the "black-box".

This is by no means a new revelation. Bit-slice devices are currently being used to perform time consuming and repetitious operations in a number of applications. For example, in the Ramtek RM-9400 Graphic Display System, bit-slice devices are used to draw primitives such as alphanumerics, vectors, images, etc. into the refresh memory.

Bit-slice architectural design is very flexible as is the firmware written to accommodate both the applications as well as the hardware. This thesis project will use a fixed bit-slice hardware and study how the firmware can be developed to achieve high speed algorithm implementation.

## D. DESCRIPTION OF THE BIT-SLICE EVALUATION BOARD

The fixed bit-slice architecture mentioned above will be the AM29203 Bit-Slice Evaluation Board. The board utilizes four AM29203 4-bit CPU slices thus giving it a 16 bit word length. The control word used in microprogramming is 48 bits in length. It is this control word that addresses each control line on the board and thereby coordinates all of the actions at each clock cycle. The board is a firmware evaluation tool and allows the user to become familiar with both the architectural details as well as the micro and the macro programming facilities available. Although the architecture is fixed, a reasonable study may still be accomplished through the evaluation of the firmware produced.

To this end, a straight forward image processing algorithm will be implemented on the board and the firmware written will demonstrate how this flexibility can be used to increase computational speed. To further emphasize the advantages, the same algorithm will be implemented using microprocessor design. A discussion will then follow to address the two results. The algorithm to be implemented will be a neighborhood averaging image smoothing routine.

## E. INTRODUCTION TO IMAGE SMOOTHING

The purpose of image smoothing is to minimize the effects of noise in the transmission channel or from poor digitization systems. Both spatial and frequency domain techniques exist to accomplish this task. In the frequency domain, this would be accomplished through the use of a low-pass filter. Spurious effects as well as the edge information exist in the high frequency part of the image. Thus, by low-pass filtering, these spurious effects are minimized, but the edge information is also altered. This causes blurring of the image. The spatial domain technique for smoothing is called neighborhood averaging. Neighborhood averaging averages those pixels closest to the point (x,y) and assigns that point the average as depicted in Figure 1. The following relation defines the process where S is the set of the coordinates of points in the neighborhood, but not including (x,y), and M is the total number of points in the set S.

$$g(x,y)=1/M \sum_{(n,m)\in S} f(m,n)$$

As a demonstration of this technique, the following experiment was performed. Using an EYECOM TV camera digitizer, a black cross on a white background was digitized and is displayed in Figure 2. The same image was then contaminated by noise and redigitized. This image is illustrated in Figure 3. The noisy image was

then smoothed by the fortran program listed under Appendix A which was executed on a VAX 11/780 under the VMS operating system. The smoothed image is illustrated in Figure 4. All illustrations were produced from a COMTAL image processor.

```
.....A.....
...B C D...
.....E.....

C=(A+B+D+E)/4
```

Figure 1
Neighborhood Averaging

The smoothed image displays much less noise than that of Figure 3, but it is also blurred as was to be expected. The image of Figure 4 was averaged a total of 7 times and took over 70 CPU minutes to complete on a VAX 11/780. Studying the program listed in Appendix A will show that some of the execution time was spent converting bytes to integers and then back to bytes in keeping with the COMTAL input format. The point is clear that such processing on a 512 X 512 pixel image (262.144 Kbytes) will take a great deal of CPU time.

It is this algorithm that will be implemented, on a much smaller data set, and then evaluated on the basis of speed enhancement.

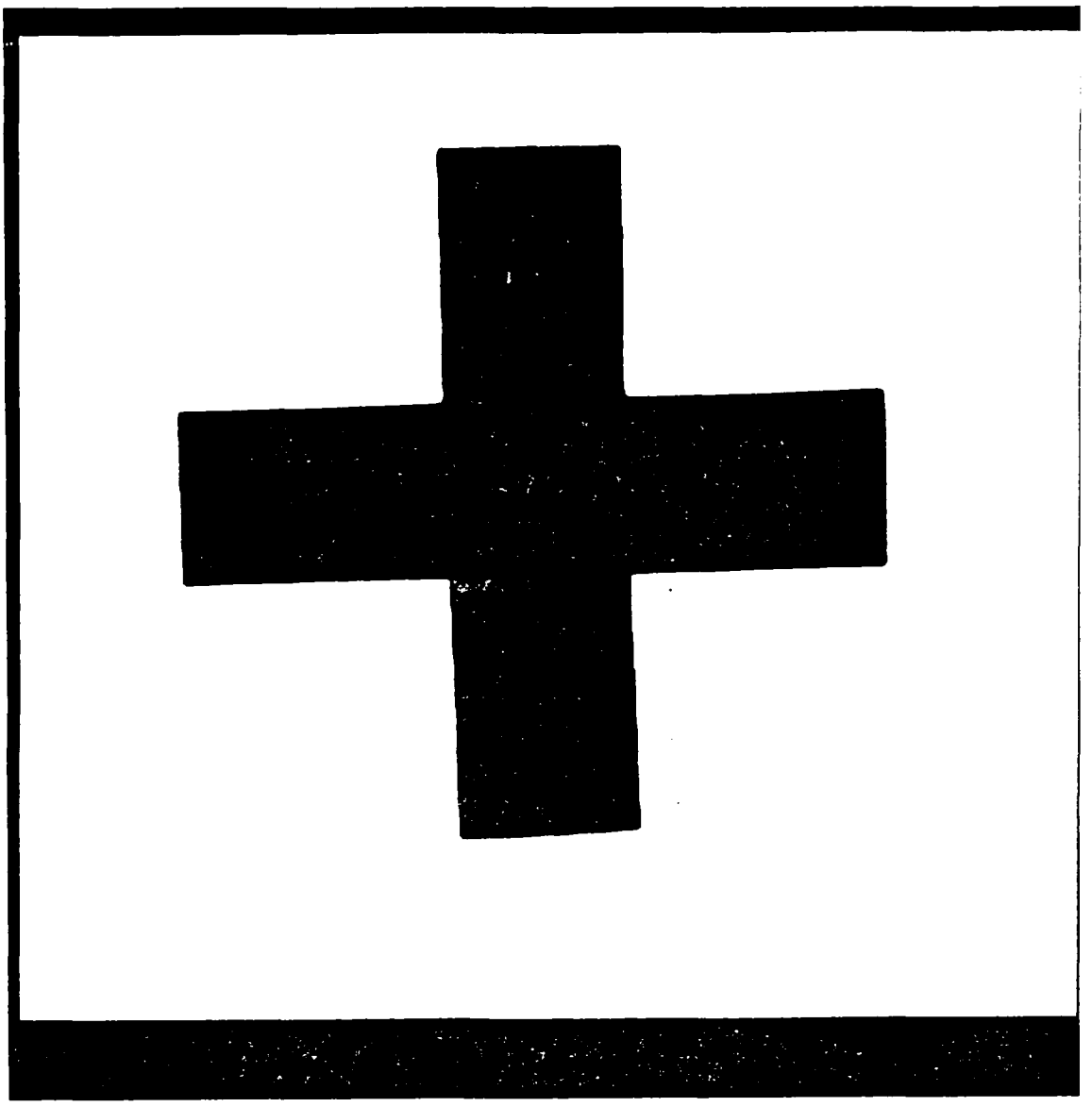


Figure 2
The Original Uncontaminated Image

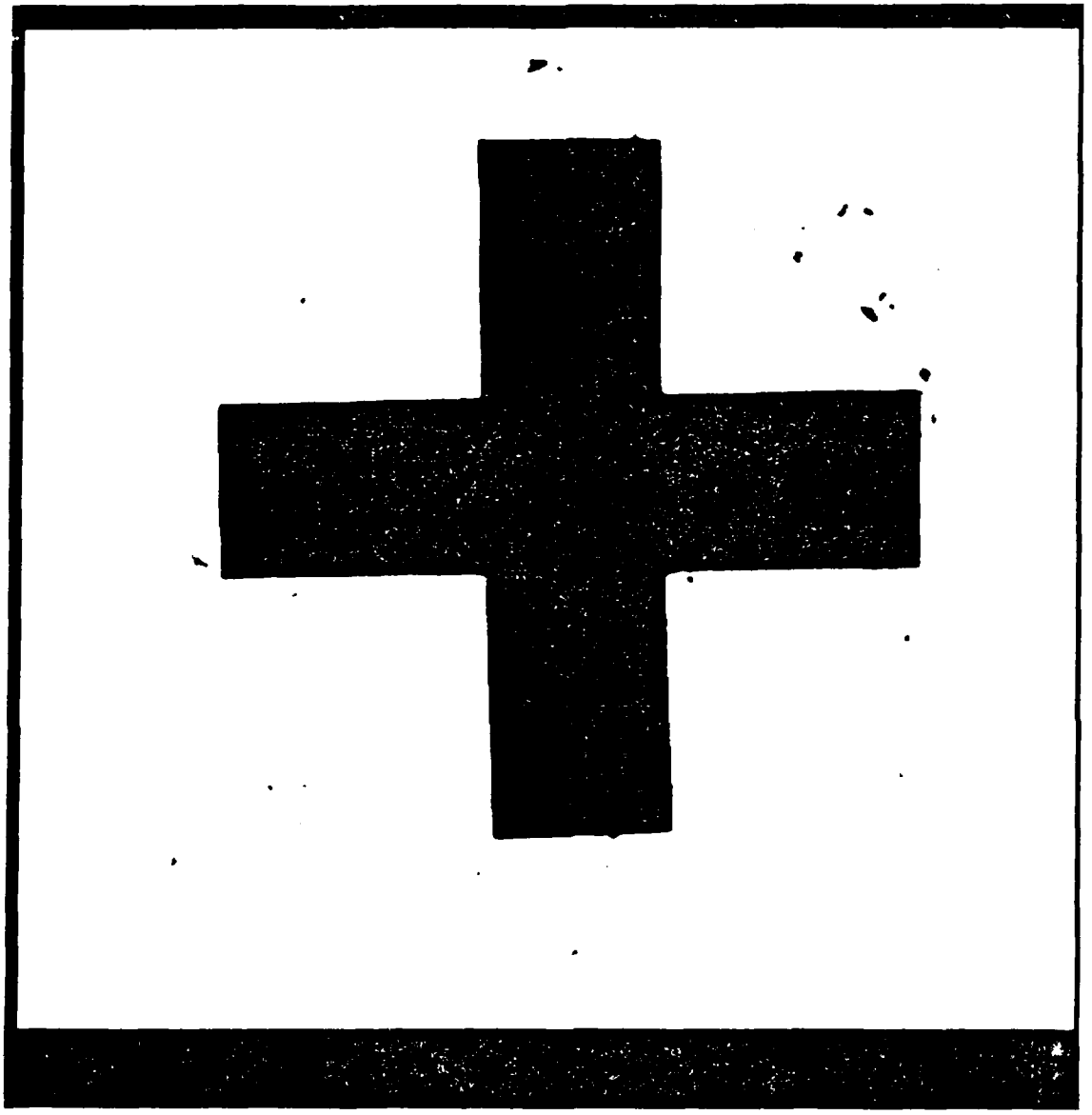


Figure 3
The Original Contaminated Image

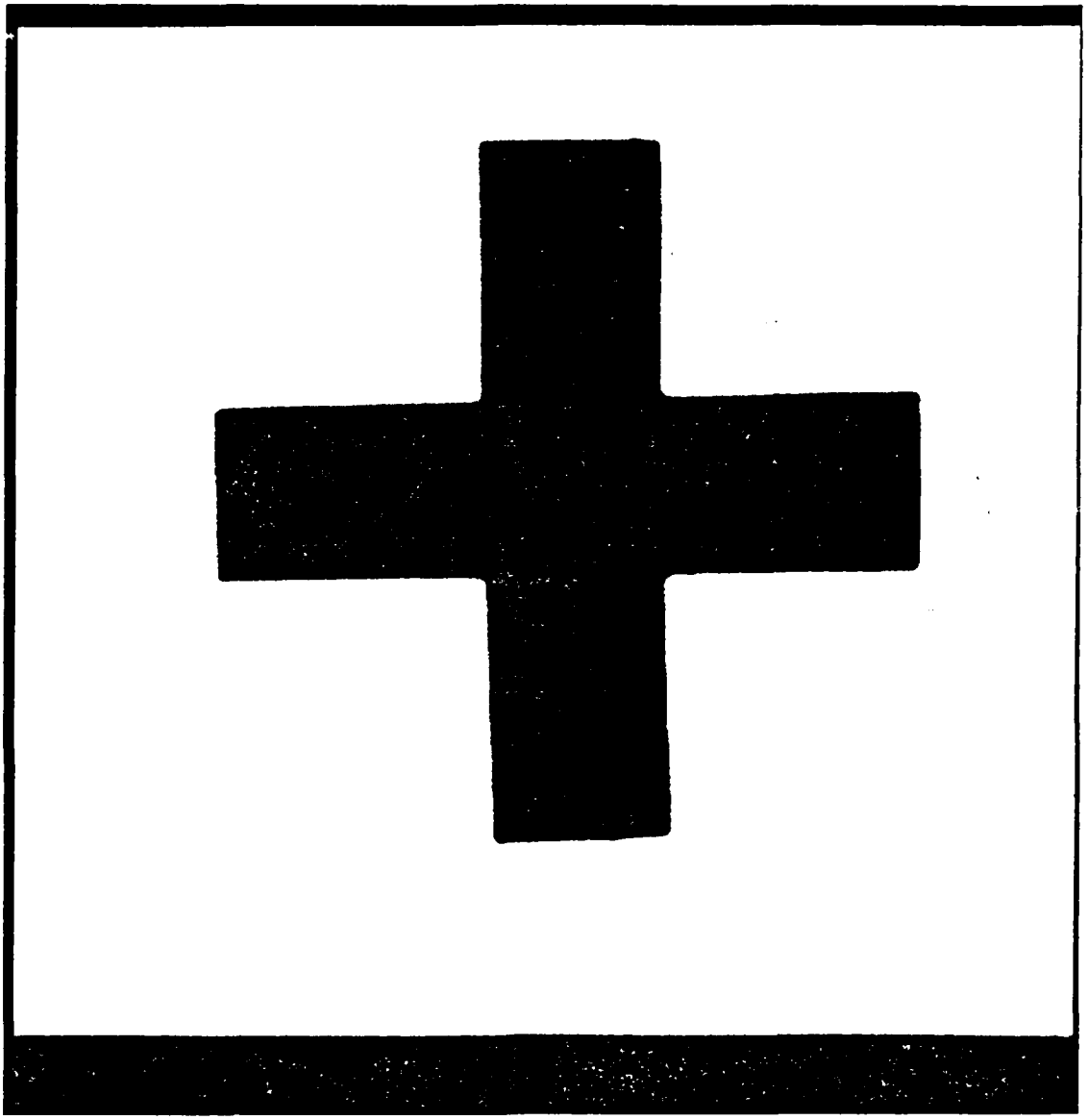


Figure 4
The Contaminated Image After Smoothing

## II. FUNCTIONAL DESCRIPTION OF THE BIT SLICE EVALUATION BOARD

### A. ARCHITECTURE

The Am29203 Bit Slice Evaluation Board actually consists of two subsystems: the evaluation board monitor software and the primary microprogrammable system. The monitor permits the user to interface to the evaluation board through the use of a terminal. In this study, emphasis is placed on the microprogrammable system's architecture and how that architecture is controlled through microprograms.

The Am29203 Evaluation Board is a fixed configuration 16-bit processor. This is accomplished through the use of four Am29203 4-bit CPU slices. A 32-bit processor could be constructed by using eight Am29203 slices demonstrating the flexibility allowed in design using a bit slice processor. The fixed architecture of the 16-bit processor is illustrated in Figure 5. This same illustration can be somewhat simplified and divided into three functional areas as shown in Figure 6. These three functional areas are the computer control unit (CCU), the arithmetic logic unit (ALU), and the memory and I/O section. Each of these

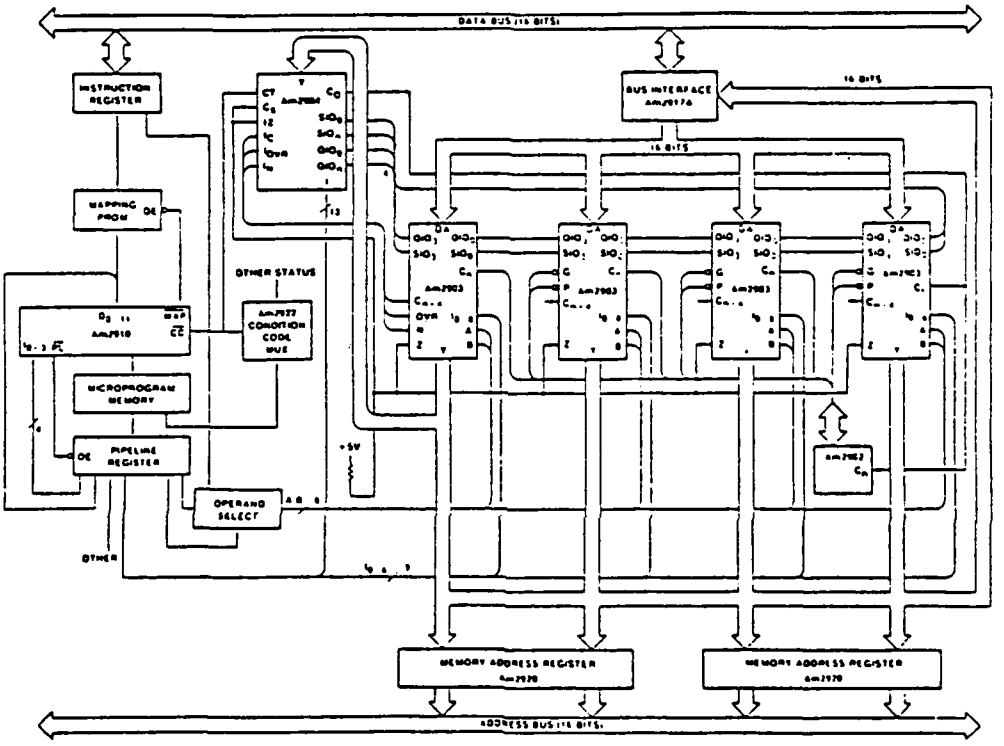



Figure 5
Evaluation Board Architecture

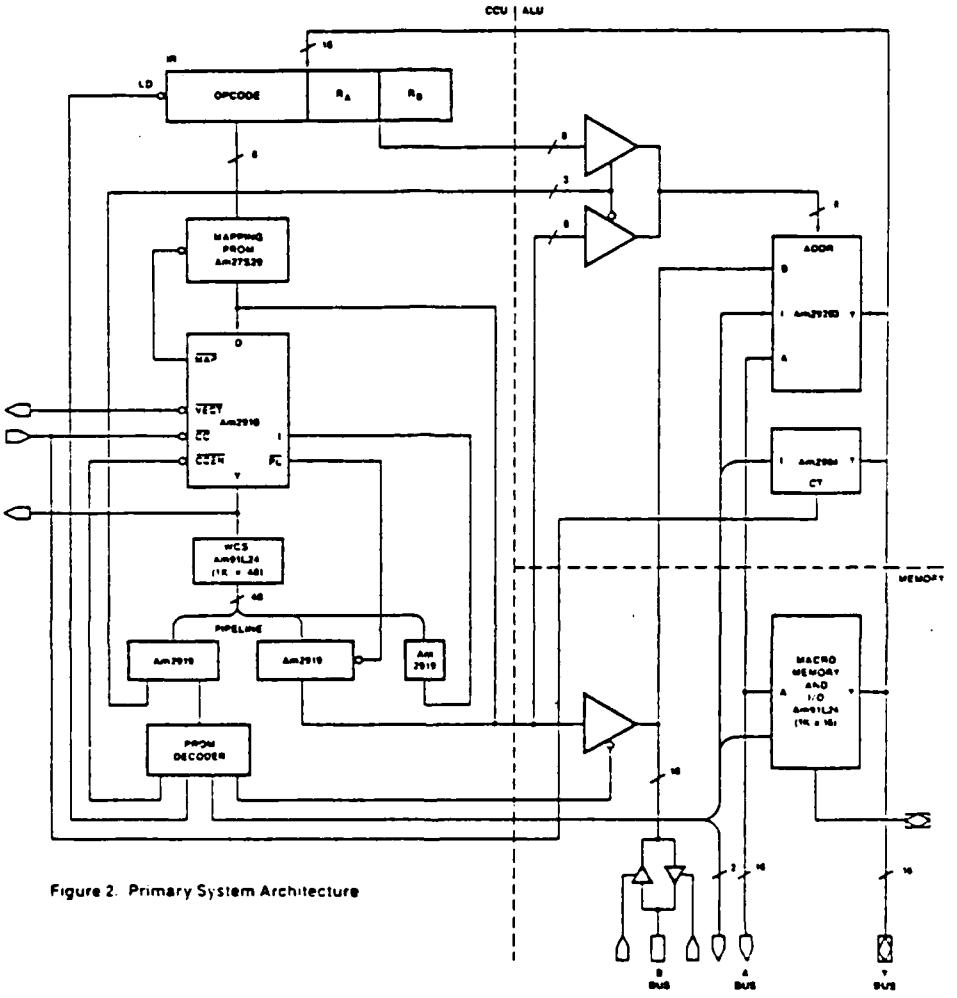



Figure 2. Primary System Architecture

Figure 6
Three Functional Areas

three functional areas will now be briefly discussed before the microprogramming of this processor is addressed.

1. The Computer Control Unit

The CCU is composed of an Am2910 sequencer which addresses 1024 words of 48-bits. These words exist in the writeable control store (WCS) RAM shown in Figure 6. These words comprise the microprogram, and the 48 bits that control each element of the 16-bit processor. The format and the utility of the 48 bit microinstruction will be discussed after the three functional areas are understood.

The pipeline register allows the microinstruction fetch to occur in parallel with the data operation. The pipeline register contains the microinstruction currently being executed. A portion of this microword instructs the Am2910 sequencer as to the address of the next microinstruction to be executed so that it is waiting at the input to the pipeline register for the next clock cycle. This has the effect of doubling the effective clock frequency.

The instruction register and the mapping PROM allow a macro-level instruction to be decoded and mapped to it's microroutine held in the WCS. Thus, an instruction set could be designed and placed into WCS.

The macro-level instructions which call these microroutines could then be used to write a macro-level program. This gives the user the ability to design his own instruction set.

This brief discussion of the CCU has demonstrated three characteristics of bit-slice design. Architectural design is flexible, architectural design allows for faster operation (pipelining), and the instruction sets are very changable rather than fixed as in microprocessor based design.

2. The Arithmetic Logic Unit

This functional area is composed of an Am29203 ALU and an Am2904 Status-and-Shift Control Unit. Figure 5 displays 4 Am2903 ALU's, but the evaluation board does employ the Am29203 ALU's. Both Figures 5 and 6 illustrate the uses of three buses. The A-bus allows the ALU output to address macro memory. The B-bus allows constants to be directly passed from the pipeline to the ALU. The Y-bus is the primary data bus for the 16-bit processor. Also, in each of the four Am29203 slices exists sixteen 4-bit registers. Since there are four Am29203 slices, there are actually sixteen 16-bit registers available to the user. Table II lists all the registers that exist within the 16-bit processor.

TABLE II
REGISTERS AVAILABLE
[Ref. 6]

| Register | Description |
|---|---|
| 0-F | Am29203 Sixteen General Purpose Registers (16 bits) |
| Q | Am29203 Q Register (16 bits) |
| I | Macroinstruction Register => IR (16 bits) |
| M | Am2904 Macro Status Register (4 bits - C,Z,N,OVR) |
| U | Am2904 Micro Status Register (4 bits - C,Z,N,OVR) |
| P | Am2910 Microprogram Counter (10 Bits) |
| R | Am2910 Register/Counter (10 bits) |
| S | Top Value On The Am2910 Internal Stack (10 bits) |

3. The Macro Memory and Input/Output

The macro memory RAM allows for the storage of 1024 16-bit words which may be machine instructions, operands or data. Thus, addresses 0000(H) - 03FF(H) on the A-bus will select the RAM location for this purpose. Placing addresses greater than 03FF(H) on the A-bus selects either I/O within the monitor section of the board or resources not available on the board. These will not be discussed.

This has been a brief architectural introduction to the resources available on the Am29203 16-bit processor board. With this in mind, the 48 bit microword used to microprogram the board will be covered next.

## B. MICROPROGRAMMING THE AM29203 EVALUATION BOARD

Figure 7 details each of the 48 bits which comprise a microinstruction. From this diagram alone it is very difficult to decipher exactly what this microword does. Only after a great amount of reading and long hours of experimentation can one hope to fully understand all of it's capabilities. Thus, a better idea would be to explain, by way of a few examples, how the 48-bit microinstruction would be written. In each of these examples, the few lines of code written will be documented in the manner recommended by the "user manual" to gain familiarity with the method.

### 1. Example 1

This example will deal with the microprogramming of the Am2910 sequencer alone. The exercise is to start at address 0000(H) with a continue instruction followed by an unconditional jump to control store location 0020(H). At address 0020(H) there is to be a continue instruction followed by a loop which executes five times. Following the loop, an unconditional subroutine call to 0200(H) will occur. At 0200(H) execute a return and jump to 0000(H) to start over again.

Figure 8 is the resulting microcode as it would appear in WCS RAM and performs the described exercise. Figure 9 details the Am2910 instruction set. These two

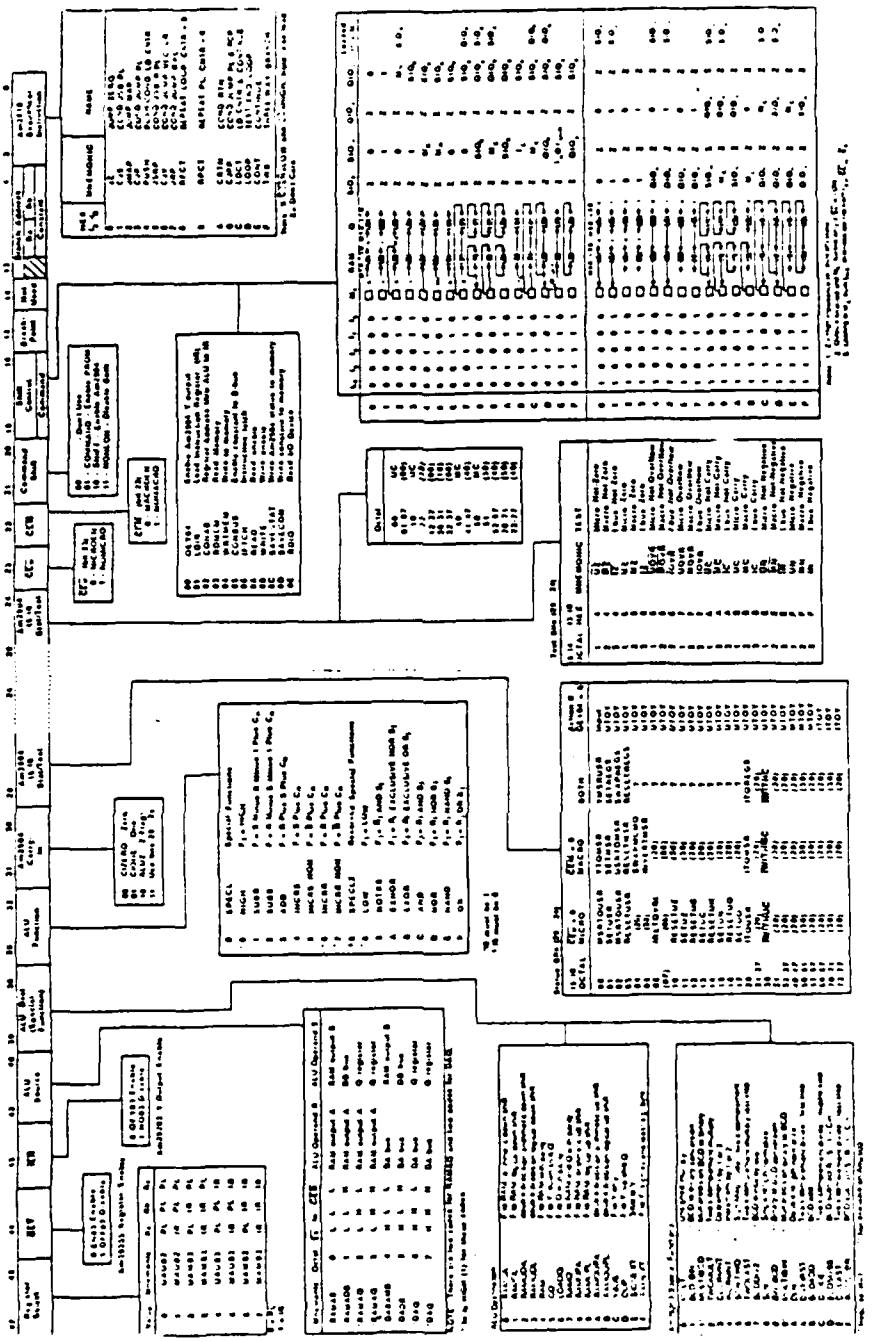


Figure 7
Microinstruction Format

figures along with the documentation in Appendix B illustrates how the Am2910 is microprogrammed and also the manner in which a microinstruction is written and documented.

2. Example 2

This example will show the use of the Am29203, i. e., the ALU. The exercise is to add the values in registers R0 and R1 and to store the result back into register R1. Figure 10 is the documentation for this single microinstruction. Careful examination of this code along with Figure 7 illustrates the use of the ALU for this simple operation.

```
0000      FFFF E4F9 FFFE  ;CONTINUE
0001      FFFF E4F9 C201  ;JUMP TO 0020(H)
 :          :    :    :     :
 :          :    :    :     :
0020      FFFF E4F9 FFFE  ;CONTINUE
0021      FFFF E4F9 C04C  ;LOOP W/ 04, CONTINUE
0022      FFFF E4F9 C229  ;REPEAT TILL CTR=0
0023      FFFF E4F9 E003  ;SUBR CALL TO 0200(H)
0024      FFFF E4F9 C001  ;JUMP TO 0000(H)
 :          :    :    :     :
 :          :    :    :     :
0200      FFFF E4F9 FFFA  ;RETURN TO 0024(H)
```

Figure 8
Microcode For Example 1

**Am2910 MICROINSTRUCTION SET.**

| HEX I3 I0 | MNEMONIC | NAME | REG/CNTR CONTENTS | FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH | | PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW | | REG/CNTR | ENABLE |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Y | STACK | Y | STACK | | |
| 0 | JZ | JUMP ZERO | X | 0 | CLEAR | 0 | CLEAR | HOLD | PL |
| 1 | CJS | COND JSB PL | X | PC | HOLD | D | PUSH | HOLD | PL |
| 2 | JMAP | JUMP MAP | X | D | HOLD | D | HOLD | HOLD | MAP |
| 3 | CJP | COND JUMP PL | X | PC | HOLD | D | HOLD | HOLD | PL |
| 4 | PUSH | PUSH/COND LD CNTR | X | PC | PUSH | PC | PUSH | Note 1 | PL |
| 5 | JSRP | COND JSB R/PL | X | R | PUSH | D | PUSH | HOLD | PL |
| 6 | CJV | COND JUMP VECTOR | X | PC | HOLD | D | HOLD | HOLD | VECT |
| 7 | JRP | COND JUMP R/PL | X | R | HOLD | D | HOLD | HOLD | PL |
| 8 | RPCT | REPEAT LOOP CNTR = 0 | ≠0 | F | HOLD | F | HOLD | DEC | PL |
| | | | =0 | PC | POP | PC | POP | HOLD | PL |
| 9 | RPCT | REPEAT PL, CNTR = 0 | ≠0 | D | HOLD | D | HOLD | DEC | PL |
| | | | =0 | PC | HOLD | PC | HOLD | HOLD | PL |
| A | CRTN | COND RTN | X | PC | HOLD | F | POP | HOLD | PL |
| B | CJPP | COND JUMP PL & POP | X | PC | HOLD | D | POP | HOLD | PL |
| C | LDCT | LD CNTR & CONTINUE | X | PC | HOLD | PC | HOLD | LOAD | PL |
| D | LOOP | TEST END LOOP | X | F | HOLD | PC | POP | HOLD | PL |
| E | CONT | CONTINUE | X | PC | HOLD | PC | HOLD | HOLD | PL |
| F | TWB | THREE WAY BRANCH | ≠0 | F | HOLD | PC | POP | DEC | PL |
| | | | =0 | D | POP | PC | POP | HOLD | PL |

Note 1: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load. X = Don't Care.

**PIN FUNCTIONS.**

| Abbreviation | Name | Function |
|---|---|---|
| $D_i$ | Direct Input Bit i | Direct input to register/counter and multiplexer. $D_0$ is LSB |
| $I_i$ | Instruction Bit i | Selects one-of-sixteen instructions for the Am2910 |
| $\overline{CC}$ | Condition Code | Used as test criterion. Pass test is a LOW on $\overline{CC}$. |
| $\overline{CCEN}$ | Condition Code Enable | Whenever the signal is HIGH, $\overline{CC}$ is ignored and the part operates as though $\overline{CC}$ were true (LOW). |
| CI | Carry-In | Low order carry input to incrementer for microprogram counter |
| $\overline{RLD}$ | Register Load | When LOW forces loading of register/counter regardless of instruction or condition |
| $\overline{OE}$ | Output Enable | Three-state control of $Y_i$ outputs |
| CP | Clock Pulse | Triggers all internal state changes at LOW-to-HIGH edge |
| VCC | +5 Volts | |
| GND | Ground | |
| $Y_i$ | Microprogram Address Bit i | Address to microprogram memory. $Y_0$ is LSB, $Y_{11}$ is MSB |
| $\overline{FULL}$ | Full | Indicates that five items are on the stack |
| $\overline{PL}$ | Pipeline Address Enable | Can select #1 source (usually Pipeline Register) as direct input source |
| $\overline{MAP}$ | Map Address Enable | Can select #2 source (usually Mapping PROM or PLA) as direct input source |
| $\overline{VECT}$ | Vector Address Enable | Can select #3 source (for example, Interrupt Starting Address) as direct input source |

Figure 9
Am2910 Instruction Set

OPERATION: R0 + R1 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#0 | ;RA=R0 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F01E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R0 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to execute the next sequential instruction.

Figure 10
Microword and Documentation for Example 2

### 3. Example 3

This example demonstrates the versatility of this processor. In one microcycle, the function 2*(R3+R4) shall be performed. Figure 11 documents this microinstruction and shows how all 48 bits of the microinstruction work together to produce the desired output.

## C. SUMMARY

This concludes a very general overview of the architectural and the microprogrammable capabilities of the Am29203 16-bit processor evaluation board. Having done this, a somewhat larger application of this processor will now be examined.

OPERATION: 2*(R3 + R4) -> R4 with R3 & R4 specified by IR

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#7 | ;reg. spec. by IR |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#8 | ;arith upshift of R4 |
| 35-32 | | FUNCT | H#3 | ;add R3 + R4 |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#20 | ;latch ALU status |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#0 | ;latch macro stat |
| 21-20 | | CMDSHFT | B#10 | ;enable Am2904 shift |
| 19-16 | | CMD | H#2 | ;upshift, zero fill |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;specified by IR |
| 7-4 | | RB | H#X | ;specified by IR |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: E083 10A2 FFFE (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be specified by the Instruction Register. The destination register, RB=R4, is enabled to be arithmetically upshifted. The Am2904 is enabled and commanded to upshift the destination register and zero fill. The registers, R3 and R4, are not specified in the pipeline in this case but instead are declared by the IR. Therefore, the contents of the IR should be 0034(H). R3 and R4 are added by the ALU and sent to R4. On their way to R4, they pass through the Am2904 and are upshifted one bit and then made ready for loading into R4 on the next rising clock edge. This upshift is equivalent to the multiplication of the result by two.

Figure 11
Microword and Documentation for Example 3

## III. BIT SLICE DEVELOPMENT OF THE IMAGE PROCESSING ALGORITHM

### A. THE ALGORITHM

The larger application previously mentioned will be a neighborhood averaging algorithm as an image smoothing technique. This algorithm will be implemented on the 5 X 5 pixel data set of bytes representing the gray scale values of the associated pixels. The image smoothing operation is defined as follows:

$$g(x,y)=1/M\sum_{(n,m)\in S} f(m,n)$$

where g(x,y) is the smoothed image, f(m,n) is the original image array and M and S are as defined in Chapter I.

Before any microprogramming can commence, some conventions must be established. The starting addresses of the original array and the smoothed array must be defined as well as how these arrays are stored in memory. The original array will be stored in macro memory starting at address 0000(H). The computed smoothed array values will be stored into macro memory starting at address 0020(H). The arrays will be stored in a row-wise manner, i.e., memory location 0003(H) will hold pixel (1,4) of the original image.

Another issue that must be addressed is what to do about the border values. The border values do not have four neighbors and thus cannot be averaged by the given definition. Thus, it was decided to simply write these values into the smoothed image as they existed in the original image.

An alternative to this would be to consider an absent neighbor to have a value of zero and to perform the averaging as defined on all the pixels including the border values. This would produce inaccurate values along the border of the image. Each time the image is successively smoothed using such a scheme, another two rows and two columns will be contaminated by these errors. This creates a propagating error affecting more and more of the border information as the image is repetitively smoothed. For this reason, the border values were chosen to be simply copied from image to image. These values will not be averaged, but will still be the values of the original image thereby avoiding the propagating contamination mentioned.

Having defined these parameters, Figure 12 diagrams how the images will exist in memory as well as how the smoothed values will be determined.

| | | | | |
|---|---|---|---|---|
| A | B | C | D | E |
| F | G | H | I | J |
| K | L | M | N | O |
| P | Q | R | S | T |
| U | V | W | X | Y |

A. Original Pixel Array

| ADDRESS | CONTENTS |
|---|---|
| 0000 | A |
| 0001 | B |
| 0002 | C |
| 0003 | D |
| 0004 | E |
| 0005 | F |
| 0006 | G |
| 0007 | H |
| 0008 | I |
| 0009 | J |
| 000A | K |
| 000B | L |
| 000C | M |
| 000D | N |
| 000E | O |
| 000F | P |
| 0010 | Q |
| 0011 | R |
| 0012 | S |
| 0013 | T |
| 0014 | U |
| 0015 | V |
| 0016 | W |
| 0017 | X |
| 0018 | Y |

B. Original Pixel Image Stored In Macro Memory

| ADDRESS | CONTENTS |
|---|---|
| 0020 | A |
| 0021 | B |
| 0022 | C |
| 0023 | D |
| 0024 | E |
| 0025 | F |
| 0026 | (B+F+H+L)/4 |
| 0027 | (C+G+I+M)/4 |
| 0028 | (D+H+J+N)/4 |
| 0029 | J |
| 002A | K |
| 002B | (G+K+M+Q)/4 |
| 002C | (H+L+N+R)/4 |
| 002D | (I+M+O+S)/4 |
| 002E | O |
| 002F | P |
| 0030 | (L+P+R+V)/4 |
| 0031 | (M+Q+S+W)/4 |
| 0032 | (N+R+T+X)/4 |
| 0033 | T |
| 0034 | U |
| 0035 | V |
| 0036 | W |
| 0037 | X |
| 0038 | Y |

C. Smoothed Pixel Image Stored In Macro Memory

Figure 12
Memory Organization for the 5 X 5 Array

## B. THE MICROROUTINE

The microroutine of Figure 13 accomplishes the algorithm outlined in the previous section. This section will only briefly address the functional details of the microroutine while the complete and detailed documentation can be found in Appendix C.

The microroutine first initiates a nested loop where the inner loop reads the four neighbors from the original array, averages the four values and writes the average to the smoothed array. This inner loop executes three times filling addresses 0026(H)-0028(H) (see Figure 12) of the smoothed array the first time through. The outer loop increments register one (R1) and register two (R2), which hold the original array addresses and the smoothed array addresses respectively, and executes the inner loop three times computing the nine averages shown in Figure 12. The remainder of the routine reads the border values from the original array and writes them to the smoothed array.

The registers hold the addresses from which data is read and stored as well as the values by which these registers must be incremented and decremented in order to complete the algorithm. Register eight (R8) is the only register that does not hold such information. It holds the outer loop counter which is decremented and tested for zero with each passing. The inner loop counter is held within the Am2910 sequencer and can be used in all looping

```
0100   LOOP1:  FFFF 3FFF C024  ;PUSH ADD. ON STACK, LD CTR W/02
0101           084F 3FD3 F14E  ;MEMORY -> R4
0102           0043 3FDF F01E  ;R1+4 -> R1
0103           084F 3FD3 F13E  ;MEMORY -> R3
0104           0043 3FDF F34E  ;R3+R4 -> R4
0105           0043 3FDF F21E  ;R1+2 -> R1
0106           084F 3FD3 F13E  ;MEMORY -> R3
0107           0043 3FDF F34E  ;R3+R4 -> R4
0108           0043 3FDF F01E  ;R1+4 -> R1
0109           084F 3FD3 F13E  ;MEMORY -> R3
010A           0043 3FDF F34E  ;R3+R4 -> R4
010B           0014 3FE0 FF4E  ;LOGICAL SHIFT RIGHT R4
010C           0014 3FE0 FF4E  ;LOGICAL SHIFT RIGHT R4
010D           00C4 3FD4 F74E  ;R4 -> MEMORY
010E           0041 3FDF F61E  ;R1-8 -> R1
010F           0044 7FFF FF7E  ;R7+1 -> R7
0110           FFFF 3FFF FFF8  ;LOOP1 (3X)
0111           0043 3FDF F21E  ;R1+2 -> R1
0112           0043 3FDF F27E  ;R7+2 -> R7
0113           0030 507F FF8E  ;R8-1 -> R8, LATCH MICROSTATREG
0114           FFFF D4D9 0003  ;IF >0, JUMP TO LOOP1 (3X)
0115           0041 3FDF F57E  ;R7-14 -> R7
0116           0041 3FDF F91E  ;R1-0F -> R1
0117   LOOP2:  FFFF 3FFF C054  ;PUSH ADD. ON STACK, LD CTR W/05
0118           084F 3FD3 F14E  ;MEMORY -> R4
0119           00C4 3FD4 F74E  ;R4 -> MEMORY
011A           0044 7FFF FF1E  ;R1+1 -> R1
011B           0044 7FFF FF7E  ;R7+1 -> R7
011C           FFFF 3FFF FFF8  ;LOOP2 (6X)
011D           F5C4 3FD2 FF8E  ;LOAD IR W/ADDRESS OF R8
011E           E544 3FD5 FC2E  ;LOAD R8 W/02
011F   LOOP3:  0043 3FDF FA1E  ;R1+3 -> R1
0120           0043 3FDF FA7E  ;R7+3 -> R7
0121   LOOP4:  FFFF 3FFF C014  ;PUSH ADD. ON STACK, LD CTR W/01
0122           084F 3FD3 F14E  ;MEMORY -> R4
0123           00C4 3FD4 F74E  ;R4 -> MEMORY
0124           0044 7FFF FF1E  ;R1+1 -> R1
0125           0044 7FFF FF7E  ;R7+1 -> R7
0126           FFFF 3FFF FFF8  ;LOOP4 (2X)
0127           0030 507F FF8E  ;R8-1 -> R8, LATCH MICROSTATREG
0128           FFFF D4D9 01F3  ;IF >0, JUMP TO LOOP3 (2X)
0129           0043 3FDF FA1E  ;R1+3 -> R1
012A           0043 3FDF FA7E  ;R7+3 -> R7
012B   LOOP5:  FFFF 3FFF C054  ;PUSH ADD. ON STACK, LD CTR W/05
012C           084F 3FD3 F14E  ;MEMORY -> R4
012D           00C4 3FD4 F74E  ;R4 -> MEMORY
012E           0044 7FFF FF1E  ;R1+1 -> R1
012F           0044 7FFF FF7E  ;R7+1 -> R7
0130           FFFF 3FFF FFF8  ;LOOP5 (6X)
0131           FFFF FFFF 7FFF  ;SET BREAKPOINT

ORIGINAL ARRAY ADDRESS:   0000-0018
AVERAGED APPAY ADDRESS:   0020-0038
INITIALIZE REGISTERS:  R0=0004; increment value
                       R1=0001; address of first neighbor read
                       R2=0002; increment value
                       R6=0008; decriment value
                       R7=0026; address first avg. stored
                       R5=0014; decriment value
                       R9=000F; decriment value
                       R8=0003; outer loop 1 counter (3X)
                       RA=0003; increment value
```

Figure 13
The 5 X 5 Microroutine

cases, but there is only one such counter on the chip. Thus, for nested loops, a separate register must be used, RB in this case.

This is only a rough overview of the process that the microroutine performs. Attention is called to Appendix C for a much more detailed explanation of the microprogramming accomplished. This documentation details how each control line is coded to execute the desired function. Comments follow each microinstruction's decomposition to further explain what the instruction does and how that effects the routine as a whole.

## C. EXECUTION TIME

In the design of a sequential processor based system, a full timing analysis must be performed for each allowable path through the system. The longest path is then used to determine the minimum clock period permissible for that design. A few alternatives exist to attempt to shorten that clock period. First, the longest path may be studied and perhaps shortened thereby allowing a shorter clock period to be assigned. The second alternative is to replace some of the system's components with faster devices also shortening the longest path. The third alternative is to use a variable clock period generator that lengthens the clock period only on those instructions necessary and reducing the clock period for

the others. This would yield an average clock period shorter than had the longest path been used as the only constraining factor on the clock period.

In order to address the execution time of the microroutine of Figure 13, both the clock period as well as the total number of microinstructions performed must be known. Careful examination of the routine shows that it performs a total of 275 microinstructions. The clock period requires a little more detailed study.

The Am29203 Evaluation Board utilizes the variable period clock generator discussed above but does not allow it to be microprogrammed, i. e., the period cannot be altered through the microword. The Am2925 clock generator and microcycle length controller permits the selection of eight different clock periods by coding pins L1, L2 and L3. These three control lines would have to be added to each microword thereby yielding a 51 bit microinstruction. The Am29203 Evaluation Board hardwires pin L2 high and pins L1 and L3 low. The crystal is configured to operate with a clock period of 51 ns or a frequency of 19.6 MHz. Table III illustrates the eight various clock periods permissible using this crystal configuration.

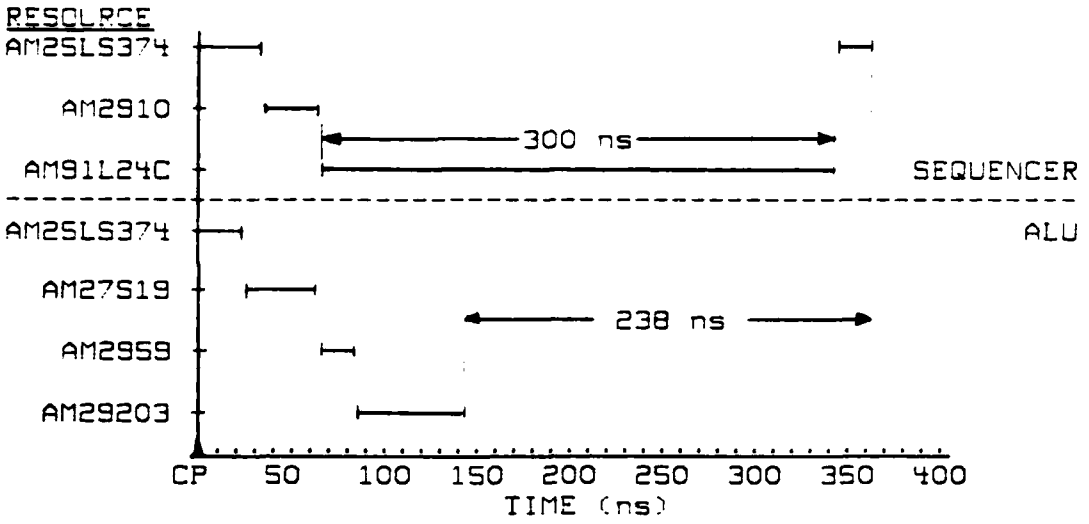
Therefore, from Table III, the Am29203 Evaluation Board is fixed to operate at approximately 2.45 MHz or with a clock period of 408 ns. The board does not allow this period to be varied.

TABLE III
PERMISSIBLE CLOCK PERIODS

| L1 | L2 | L3 | Clock Period (P=51 ns) |
|---|---|---|---|
| 0 | 0 | 0 | 3P (153 ns) |
| 0 | 0 | 1 | 4P (204 ns) |
| 0 | 1 | 0 | 8P (408 ns) |
| 0 | 1 | 1 | 7P (357 ns) |
| 1 | 0 | 0 | 10P (510 ns) |
| 1 | 0 | 1 | 5P (255 ns) |
| 1 | 1 | 0 | 9P (459 ns) |
| 1 | 1 | 1 | 6P (306 ns) |

The microroutine of Figure 13 performs 275 microinstructions, each taking 408 ns, yielding an execution time of 112.2 microseconds. As previously mentioned, there are some alternatives to study in order to reduce this execution time. The paths are fixed by the microroutine, but improvements can be made in both the speed of the devices used as well as varying the period of the clock generator.

Every instruction executed on the Evaluation Board must either be found in macro memory or in writeable control store (WCS). In either case, or path, a read from RAM is required. This read consumes 300 ns of execution time for the Am91L24. Reading from RAM is always a costly operation and in this case, is the most costly on the board. For instance, in order to add two registers using the ALU and perform a simple "continue" operation with the sequencer, the timing path is as shown in Figure 14.




AM2910 Time Path

| | | |
|---|---|---|
| Clock-To-Output of Pipeline (AM2SLS374) | : | After CP, time before the instruction leaves the pipeline register. (28 ns) |
| I-To-Y of AM2910 | : | Time AM2910 takes to decipher the instruction received and to output next instruction address to WCS (35 ns) |
| Address-To-Output Of WCS (AM91L24C) | : | Time AM9124C takes to output next instruction to the pipeline. (300 ns) |
| Set Up On Pipeline (AM2SLS374) | : | Time to set up next instruction on the input to the pipeline register. (20 ns) |

AM29203 Time Path

| | | |
|---|---|---|
| Clock-To-Output Of Pipeline (AM2SLS374) | : | After CP, time before the instruction leaves the pipeline register. (28 ns) |
| Address-To-Output Of Command ROM (AM27S19) | : | Time needed to decode received command into ALU control codes. (35 ns) |
| Enable-To-Output Of Control Gates(AM2959) | : | Time ALU codes need to reach the ALU. (15 ns) |
| DB-To-Y Of ALU (AM29203) | : | Time ALU needs to execute the function. (58 ns) |
| Set Up Y On Result Register (AM29203) | : | Time that result must set up on Y-bus for loading into the register at the rising clock edge. (17 ns) |

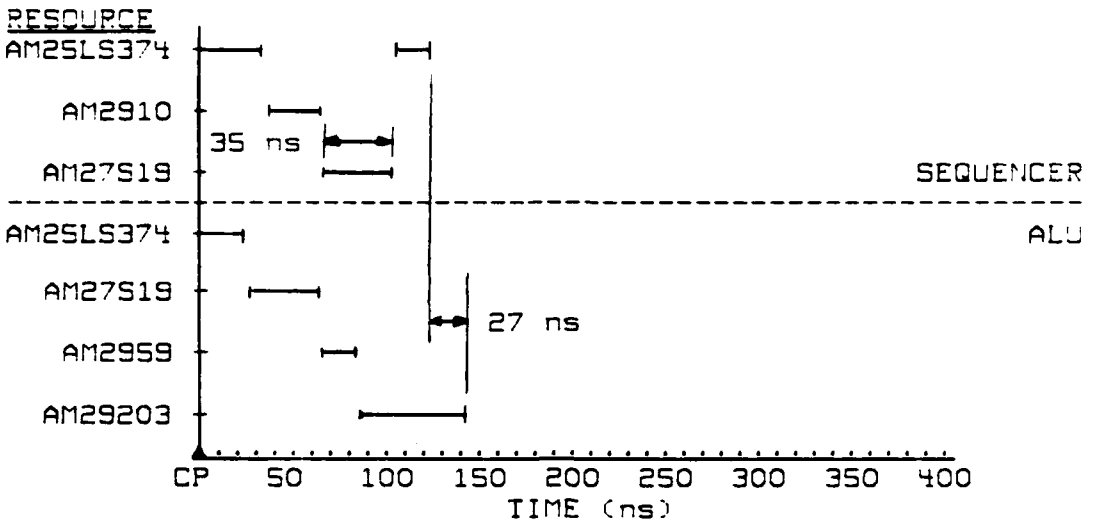
Figure 14
Time Path Illustration with RAM as WCS

Since RAM must be addressed for each instruction, this is the constraining factor in determining the clock period. I can only assume that this is why the board is fixed to run with a constant clock period There is no reason to vary the clock period when each instruction requires this 300 ns overhead. All instructions will take very nearly 400 ns regardless of there complexity.

In order to speed up the routine at hand, that RAM in WCS will have to be removed. The ALU is idle for 238 ns while the sequencer is determining the next instruction to be executed. This is a wasteful use of the resources.

Since the microroutine is fixed and there is no reason to update it, it can be written to a fast PROM. The WCS RAM can then be replaced with this PROM. Such PROM's can have address-to-output times as fast as 35 ns. With such an improvement, the same timing path previously detailed could be reduced to 118 ns which is a 69% reduction. The idle time is now only 27 ns, down from 238 ns. The resources, the ALU and the sequencer, are both being used more efficiently after this change is made. Figure 15 illustrates this reduction in the timing path.

The microroutine still reads data from the original array and stores the averaged data to the smoothed array and both reside in RAM. Therefore, the timing path for these operations will still exceed 300 ns. Figure 16 illustrates the timing path for a memory read and data



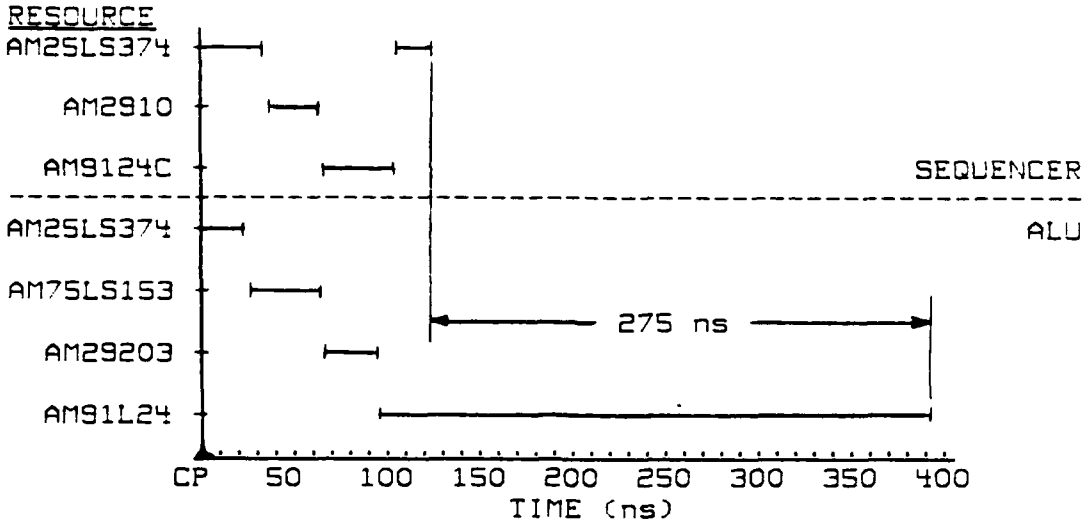

AM2910 Time Path

| | | |
|---|---|---|
| Clock-To-Output of Pipeline (AM25LS374) | : | After CP, time before the instruction leaves the pipeline register. (28 ns) |
| I-To-Y of AM2910 | : | Time AM2910 takes to decipher the instruction received and to output next instruction address to WCS (35 ns) |
| Address-To-Output Of PROM (AM27S19) | : | Time AM27S19 takes to output next instruction to the pipeline. (35 ns) |
| Set Up On Pipeline (AM25LS374) | : | Time to set up next instruction on the input to the pipeline register. (20 ns) |

AM29203 Time Path

| | | |
|---|---|---|
| Clock-To-Output Of Pipeline (AM25LS374) | : | After CP, time before the instruction leaves the pipeline register. (28 ns) |
| Address-To-Output Of Command ROM (AM27S19) | : | Time needed to decode received command into ALU control codes. (35 ns) |
| Enable-To-Output Of Control Gates(AM2959) | : | Time ALU codes need to reach the ALU. (15 ns) |
| DB-To-Y Of ALU (AM29203) | : | Time ALU needs to execute the function. (58 ns) |
| Set Up Y On Result Register (AM29203) | : | Time that result must set up on Y-bus for loading into the register at the rising clock edge. (17 ns) |

Figure 15
Time Path Illustration after
PROM Substitution for WCS RAM




AM2910 Time Path

See Figure XX for a detailed explanation of this path.

AM29203 Time Path

| | | |
|---|---|---|
| Clock-To-Output Of Pipeline (AM2S1S374) | : | After CP, time before the instruction leaves the pipeline register. (28 ns) |
| A-B MUX Select To Output (AM7SLS1S3) | : | Time needed by MUX to set R4 to receive data from memory and to set R7 as the address of the data. (38 ns) |
| Address-To-DA (AM29203) | : | Time to place memory address on A-bus from the ALU. (30 ns) |
| Memory-To-Y-bus (AM91L24) | : | Time after address is received by RAM and data is output to the Y-bus. (280 ns) |
| Set Up Of Data On Register (AM91L24) | : | Time that data must set up on R4 for loading on the rising edge of the next clock pulse. (17 ns) |

Figure 16
Time Path Illustration with
Data RAM Access

load into R4 yielding an execution time of 393 ns. With the replacement of the WCS RAM with the PROM, no longer are all instructions in need of the extra 300 ns. This is where the application of the variable period clock generator will be most handy. For those instructions not addressing RAM, a much shorter clock period can be used than for those instructions addressing RAM. The only cost for varying the clock period will be to add 3 bits to the microword lengthening it to 51 bits. Therefore, the PROM which replaced the WCS RAM will need to be at least 51 bits wide. Considerable savings will be realized in execution time since the routine executes only 77 such RAM operations and 198 of the shorter operations. Table IV illustrates the instruction time analysis.

TABLE IV
INSTRUCTION TIMING ANALYSIS

| Instruction Type | Read/Write To RAM | No Read/Write To Ram |
|---|---|---|
| Execution Time | 383 ns | 118 ns |
| Percentage Of Instruction Stream | 28% | 72% |
| 19.6 MHz P=51 ns | 8P (408 ns) | 3P (153 ns) |

The average microcycle time of the variable clock cycle generator with the WCS RAM replaced by the PROM is calculated as follows:

$$(0.28 \times 408) + (0.72 \times 153) = 224.4 \text{ ns}$$

This is a 45% increase in system performance simply by replacing the WCS RAM with a fast PROM and increasing the microword bit length to incorporate two kinds of clock periods rather than the single clock period of 408 ns. In fact, close inspection of Table III shows that to code the clock generator to run at 3P and 8P only requires the switching of pin L2 with pins L1 and L3 tied to ground. This means that the microword only needs to be lengthened to 49 bits rather than the 51 bits mentioned earlier. Further timing analysis could reveal more then two distinctly different time paths and thus enable the coding of the clock generator to further reduce the average clock period.

After these improvements, the microroutine will execute in 61.71 microseconds, down from 112.2 microseconds.

## D. THE 512 X 512 ARRAY

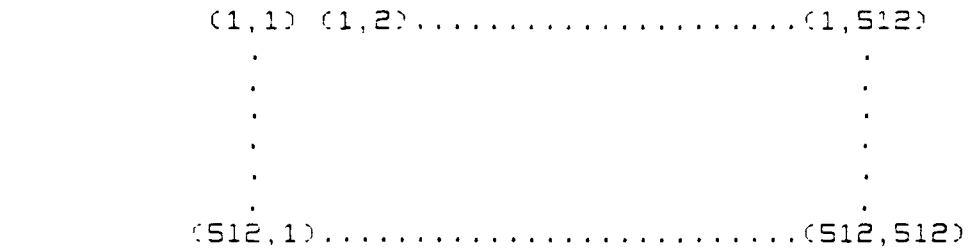
Having defined the algorithm and the microroutine for a small 5 X 5 array, their application to a more realistic digital image data array will now be discussed.

If a 512 X 512 image data array is to be smoothed, both the architecture as well as the microroutine will have to be modified. A 512 X 512 pixel array alone would reside in 264.144 kilobytes of memory assuming that 256 gray scales are to be used thus enabling the gray scale

data to be represented by a byte. There must also be the same amount of memory available to store the smoothed array. In order to address the total 524.288 kilobytes of memory proposed, the address bus to macro memory must be 20 bits wide. Since registers hold memory locations and these memory addresses are incrimented and decrimented by the ALU, the registers as well as the data bus must be 20 bits wide. The macro memory address bus, registers and data bus are presently only 16 bits wide. The addition of one Am29203 4-bit ALU slice and additional wiring onto the data and address buses would accomplish this. Since sixteen 4 bit registers reside in each Am29203, the addition of one yields sixteen 20 bit registers as needed.

The microroutine itself needs little changing except for the address incriment and decriment values. Figure 17 illustrates what values would need to be changed for the routine to operate on a 512 X 512 pixel array and Figure 18 is the updated microroutine that would operate on a 512 X 512 array.

This updated microroutine has neither been documented nor has it been tested. The routine's looping parameters have only been changed. The operations are identical and the average microcycle time should be the same as that for the 5 X 5 array.




A. Original 512 X 512 Pixel Array

| ADDRESS | CONTENTS |
|---|---|
| 00000 | (1,1) |
| 00001 | (1,2) |
| : | : |
| 001FF | (1,512) |
| 00200 | (2,1) |
| 00201 | (2,2) |
| 00202 | (2,3) |
| : | : |
| 003FF | (2,512) |
| 00400 | (3,1) |
| 00401 | (3,2) |
| : | : |
| : | : |
| : | : |
| 3FFFF | (512,512) |

B. Original 512 X 512 Pixel Image Stored In Macro Memory

| ADDRESS | CONTENTS |
|---|---|
| 40000 | (1,1) |
| 40001 | (1,2) |
| : | : |
| 401FF | (1,512) |
| 40200 | (2,1) |
| 40201 | (1,2)+(2,1)+(2,3)+(3,2)/4 |
| 40202 | (1,3)+(2,2)+(2,4)+(3,3)/4 |
| : | : |
| 403FF | (2,512) |
| 40400 | (3,1) |
| 40401 | (2,2)+(3,1)+(3,3)+(4,2)/4 |
| : | : |
| : | : |
| : | : |
| 7FFFF | (512,512) |

C. Smoothed 512 X 512 Pixel Image Stored In Macro Memory

Figure 17
Memory Organization for the 512 X 512 Array

```
0100   LOOP1: 0 FFFF 3FFF 0F4D  ;PUSH ADD. ON STACK, LD CTR W/1F0
0101           1 084F 3FD3 F14E  ;MEMORY -> R4
0102           0 0043 3FDF F01E  ;R1+511 -> R1
0103           1 084F 3FD3 F13E  ;MEMORY -> R3
0104           0 0043 3FDF F34E  ;R3+R4 -> R4
0105           0 0043 3FDF F21E  ;R1+2 -> R1
0106           1 084F 3FD3 F13E  ;MEMORY -> R3
0107           0 0043 3FDF F34E  ;R3+R4 -> R4
0108           0 0043 3FDF F01E  ;R1+511 -> R1
0109           1 084F 3FD3 F13E  ;MEMORY -> R3
010A           0 0043 3FDF F34E  ;R3+R4 -> R4
010B           0 0014 3FE0 FF4E  ;LOGICAL SHIFT RIGHT R4
010C           0 0014 3FE0 FF4E  ;LOGICAL SHIFT RIGHT R4
010D           1 00C4 3FD4 F74E  ;R4 -> MEMORY
010E           0 0041 3FDF F61E  ;R1-3FE -> R1
010F           0 0044 7FFF FF7E  ;R7+1 -> R7
0110           0 FFFF 3FFF FFF8  ;LOOP1 (3X)
0111           0 0043 3FDF F21E  ;R1+2 -> R1
0112           0 0043 3FDF F27E  ;R7+2 -> R7
0113           0 0030 507F FF8E  ;R8-1 -> R8, LATCH MICROSTATREG
0114           0 FFFF D4D9 0003  ;IF >0, JUMP TO LOOP1 (3X)
0115           0 0041 3FDF F57E  ;R7-3FE00 -> R7
0116           0 0041 3FDF F91E  ;R1-3FC00 -> R1
0117   LOOP2: 0 FFFF 3FFF E014  ;PUSH ADD. ON STACK, LD CTR W/201
0118           1 084F 3FD3 F14E  ;MEMORY -> R4
0119           1 00C4 3FD4 F74E  ;R4 -> MEMORY
011A           0 0044 7FFF FF1E  ;R1+1 -> R1
011B           0 0044 7FFF FF7E  ;R7+1 -> R7
011C           0 FFFF 3FFF FFF8  ;LOOP2 (6X)
011D           0 F5C4 3FD2 FF8E  ;LOAD IR W/ADDRESS OF R8
011E           0 E544 3FD5 0FDE  ;LOAD R8 W/1FD
011F   LOOP3: 0 0043 3FDF FA1E  ;R1+1FE -> R1
0120           0 0043 3FDF FA7E  ;R7+1FE -> R7
0121   LOOP4: 0 FFFF 3FFF C014  ;PUSH ADD. ON STACK, LD CTR W/01
0122           1 084F 3FD3 F14E  ;MEMORY -> R4
0123           1 00C4 3FD4 F74E  ;R4 -> MEMORY
0124           0 0044 7FFF FF1E  ;R1+1 -> R1
0125           0 0044 7FFF FF7E  ;R7+1 -> R7
0126           0 FFFF 3FFF FFF8  ;LOOP4 (2X)
0127           0 0030 507F FF8E  ;R8-1 -> R8, LATCH MICROSTATREG
0128           0 FFFF D4D9 01F3  ;IF >0, JUMP TO LOOP3 (2X)
0129           0 0043 3FDF FA1E  ;R1+1FE -> R1
012A           0 0043 3FDF FA7E  ;R7+1FE -> R7
012B   LOOP5: 0 FFFF 3FFF E014  ;PUSH ADD. ON STACK, LD CTR W/201
012C           1 084F 3FD3 F14E  ;MEMORY -> R4
012D           1 00C4 3FD4 F74E  ;R4 -> MEMORY
012E           0 0044 7FFF FF1E  ;R1+1 -> R1
012F           0 0044 7FFF FF7E  ;R7+1 -> R7
0130           0 FFFF 3FFF FFF8  ;LOOP5 (6X)
0131           0 FFFF FFFF 7FFF  ;SET BREAKPOINT

ORIGINAL ARRAY ADDRESS:   00000-3FFFF
AVERAGED ARRAY ADDRESS:   40000-7FFFF
INITIALIZE REGISTERS:  R0-001FF; increment value
                       R1-00001; address of first neighbor read
                       R2-00002; increment value
                       R6-003FE; decrement value
                       R7-40201; address first avg. stored
                       R5-3FE00; decrement value
                       R9-3FC00; decrement value
                       R8-001FE; outer loop 1 counter (510X)
                       RA-001FE; increment value
```

Figure 18
The 512 X 512 Microroutine

The microroutine of Figure 18 performs 4,438,056 microinstructions. Assuming that the routine resides on PROM and that the clock generator is varied as the routine shows, the average microcycle time of 224.4 ns found in the 5 X 5 case should hold true in this case. Therefore, the execution time of this routine should be approximately one second.

# IV. MICROPROCESSOR DEVELOPMENT OF THE IMAGE PROCESSING ALGORITHM

## A. THE CODE AND EXECUTION TIME

For comparative purposes, the same 5 X 5 data array averaging algorithm previously developed is implemented using a Z-80 microprocessor. Architectural and instruction set constraints are discussed and compared to those of bit slice development within the context of execution speed.

The assembly language routine is displayed in Figure 19. This algorithm is identical in operation to the bit slice routine.

In order to address the execution time of the routine of Figure 19, the clock period and the number of microinstructions performed must be known. These microinstructions are called T states. Since the Z-80 cannot be microprogrammed by the user, each assembly level instruction must be fetched from memory, decoded and then executed. A fetch will require three microcycles (T states) to retreive the instruction from memory and to load it into the instruction register. The decode phase will require one T state to decode the assembly level instruction into it's microroutine's starting address. The microroutine is then executed to it's completion.

```
                                        ORG     0200H
0279                            AMAT    EQU     ENDADD+1
0297                            BMAT    EQU     ENDADD+31
0278                            AAMAT   EQU     ENDADD
0291                            BBMAT   EQU     ENDADD+25
0200    DD 21 0279                      LD      IX,AMAT
0204    FD 21 0297                      LD      IY,BMAT
0208    0E 03                           LD      C,3
020A    06 03                   LOOP1:  LD      B,3
020C    3E 00                   LOOP2:  LD      A,0
020E    DD 86 00                        ADD     A,(IX+0)
0211    DD 86 04                        ADD     A,(IX+4)
0214    DD 86 06                        ADD     A,(IX+6)
0217    DD 86 0A                        ADD     A,(IX+10)
021A    CB 3F                           SRL     A
021C    CB 3F                           SRL     A
021E    FD 77 00                        LD      (IY+0),A
0221    DD 23                           INC     IX
0223    FD 23                           INC     IY
0225    05                              DEC     B
0226    20 E4                           JR      NZ,LOOP2
0228    DD 23                           INC     IX
022A    DD 23                           INC     IX
022C    FD 23                           INC     IY
022E    FD 23                           INC     IY
0230    0D                              DEC     C
0231    20 D7                           JR      NZ,LOOP1
0233    DD 21 0278                      LD      IX,AAMAT
0237    FD 21 0291                      LD      IY,BBMAT
023B    0E 06                           LD      C,6
023D    DD 7E 00                LOOP3:  LD      A,(IX+0)
0240    FD 77 00                        LD      (IY+0),A
0243    DD 23                           INC     IX
0245    FD 23                           INC     IY
0247    0D                              DEC     C
0248    20 F3                           JR      NZ,LOOP3
024A    11 0003                         LD      DE,3
024D    0E 02                           LD      C,2
024F    DD 19                   LOOP4:  ADD     IX,DE
0251    FD 19                           ADD     IY,DE
0253    06 02                           LD      B,2
0255    DD 7E 00                LOOP5:  LD      A,(IX+0)
0258    FD 77 00                        LD      (IY+0),A
025B    DD 23                           INC     IX
025D    FD 23                           INC     IY
025F    05                              DEC     B
0260    20 F3                           JR      NZ,LOOP5
0262    0D                              DEC     C
0263    20 EA                           JR      NZ,LOOP4
0265    DD 19                           ADD     IX,DE
0267    FD 19                           ADD     IY,DE
0269    0E 06                           LD      C,6
026B    DD 7E 00                LOOP6:  LD      A,(IX+0)
026E    FD 77 00                        LD      (IY+0),A
0271    DD 23                           INC     IX
0273    FD 23                           INC     IY
0275    0D                              DEC     C
0276    20 F3                           JR      NZ,LOOP6
0278                            ENDADD: END
```

Figure 19
The Z-80 Assembly Language Routine

It appears that for each assembly level instruction performed, there is a four microcycle overhead before execution can begin. This is not always true. In some, but not all, cases, the fetch of the next instruction can occur while the current instruction is executing. This fetch-execute overlap can save two microcycles per instruction if the buses or facilities are available without conflict. The control unit knows whether an overlap is possible. For comparitive purposes, a worst case analysis will be used, i.e., it is assumed that no fetch-execute overlaps will occur. With this assumption being made, the routine of Figure 19 executes 2505 microcycles. Using a 2 MHz clock, the Z-80's microcycle is 500 ns long. The routine then executes in 1.2525 milliseconds. This can be compared to the execution time of 61.71 microseconds in the bit slice case. Therefore, the Z-80 routine executes 20 times slower than the bit slice routine.

This comparison is made having assumed a worst case execution time for the Z-80. Another consideration is that the bit slice processor uses a 16 bit data bus while the Z-80 uses only an 8 bit bus. Nevertheless, as many as 4 microcycles are being wasted in fetching and decoding the instruction prior to execution. The price paid to

avoid this fetch/decode overhead in the bit slice case is coding complexity. Assembly language allows the programmer to take a much higher level look at the instructions he wishes to perform while the microinstruction programmer is burdened with coding each control line and being careful not to access resources already in use. In many cases, the shortened programming time is well worth the loss in architectural and instructional flexibility.

For the 5 X 5 case, the Z-80 internal registers, 8 or 16 bits wide, and data bus widths, 8 bits, were adequate. Expansion to the 20 bit registers and 20 bit data bus widths needed for the 512 X 512 data array averaging would not be possible. Even the use of a 16 bit microprocessor would not be wide enough. This deficiency would have to be handled through the writing of a more complex software routine which would allow register constants to be held in more than one noncontiguous register. This more complex routine would require even more time to execute, while in the bit slice case, architectural changes could be accomodated. The same routine could be used as long as the register values are updated to match the size of the array being averaged.

# V. CONCLUSIONS AND RECOMMENDATIONS

## A. EVALUATION BOARD SHORTCOMINGS

The Am29203 bit slice evaluation board has proved to be a very instructive tool. It allows the user to study the characteristics of bit slice microprocessor design. Although the hardware is fixed, the user must come to understand the interrelationships of the various bit slice devices before any microprogramming can begin. In doing so, it becomes evident that such architecture is adaptable to a variety of applications. Bus widths are flexible and the instructions sets can be readily adapted to such changes in the architecture.

The availability of a variable period clock generator on the board is a nice feature but it is fixed to run at a constant frequency. This hardwiring of the clock generator is understandable considering the long timing paths associated with reading from RAM, but the WCS RAM may be replaced by a PROM thus reducing the timing path a great deal in some cases. The user manual describes the replacement of the WCS RAM with a PROM which has the designed microcode burned into it as a desirable feature. Such replacement would be more attractive if the board allowed for the three additional control lines that could

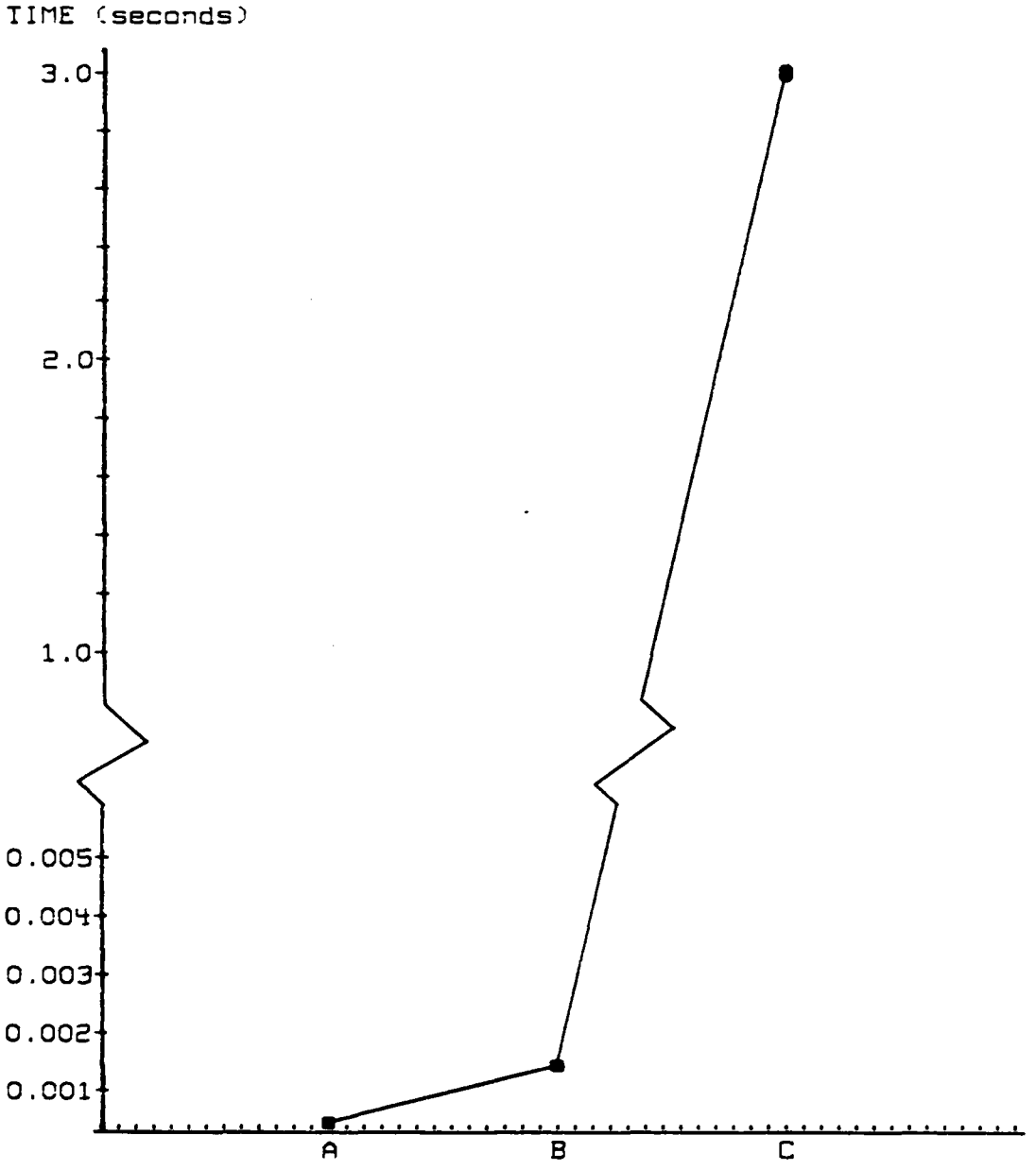
be coded to vary the clock period and take advantage of the shorter time paths resulting. This would improve the throughput of the system. The user manual neither mentions that the clock generator is fixed to a period of 408 ns nor does it mention any means by which to vary this period. Such flexibility is an attractive characteristic of bit slice design but the board completely ignores it. Nevertheless, the user should be aware of this feature as a design parameter.

The monitor built into the evaluation board is very useful. It permits the loading of all registers and the loading of microcode or macrocode. It also allows the code to be executed one line at a time or executed all at once. The monitor does not allow for the interface of any mass storage or hard copy peripherals though. The user manual does recommend that the board be interfaced to a host computer. The details of such an interface are briefly mentioned in the manual and a program written in C is also included which permits the uploading and the downloading of code from and to the computer's disk drive. Such code could then be stored for later retrieval or printed as desired. This is a very attractive feature and is highly recommended for any future endeavors larger than the microroutine written here within. For small applications, the monitor included with the system and a hard terminal will suffice.

## B. DISCUSSION OF BIT SLICE DESIGN

The execution speed of a register transfer language is very fast when compared to the speeds of the same routine written in an assembly language and in FORTRAN. For the image smoothing of a 5 X 5 array, the FORTRAN routine executed in approximately 3 CPU seconds on the VAX 11/780. That 3 seconds includes the multi-user system software overhead. The assembly language routine was estimated to run in 1.2525 milliseconds and the bit slice microroutine executes in 61.71 microseconds. Figure 20 depicts these execution times graphically. Clearly, the bit slice design has yielded a very fast processor.

Such an execution time comparison has ignored an equally important time feature, namely, design time. The bit slice microroutine was by far the fastest in execution but was also by far the most difficult to write. The opposite is true for the FORTRAN program. The reason for this vast difference in the time needed to write the code is the difference in the level of understanding of the underlying hardware required. The writing of the FORTRAN program required virtually no understanding of the hardware it was to be run on. The bit slice microroutine, on the other hand, required a full and detailed understanding of the hardware it was to be run on. This




A: Bit Slice Execution Time
B: Microprocessor Execution Time
C: FORTRAN Execution Time

Figure 20
Comparison of Execution Times

learning overhead is very costly. Therefore, more than execution time alone must be considered when selecting a type of design.

Should the routine be needed for only a few applications, the design time saved by writing the routine in a high level language, such as FORTRAN, becomes the dominating factor. If the routine is to be executed a great number of times, the large investment in time paid in performing bit slice design will be refunded by way of the savings in the execution time. Microprocessor design lies somewhere between these two concerns.

Image processing lends itself well to bit slice design. Such algorithms need to be executed on the massive amounts of digital image data being gathered in today's world of satelite imagery. The price paid in long design times will be quickly returned with interest when the savings in execution time are considered. Image smoothing is just one such algorithm and was used here to illustrate the savings in execution time compared to other design choices. The result is that the application of bit slice design to digital image processing systems is a very attractive alternative.

# APPENDIX A
## FORTRAN IMAGE SMOOTHING PROGRAM

```
C          THE PURPOSE OF THIS PROGRAM IS TO CONVERT THE
C          640 X 480 PIXEL IMAGE PRODUCED BY THE EYECOM
C          DIGITIZER TO THE 512 X 512 PIXEL IMAGE NECESSARY
C          FOR DISPLAY ON THE COMTAL UNIT.  THE IMAGE IS
C          THEN SMOOTHED THROUGH THE USE OF NEIGHBORHOOD
C          AVERAGING.  THE IMAGE TO BE SMOOTHED IS IN THE
C          FILE N.DAT.  THE SMOOTHED IMAGE IS THEN STORED
C          INTO THE FILE S.DAT.
C
           BYTE A(512,512),B(512,512),C(4)
           INTEGER M,N,P,I,J,Z,ZZ,CC(4)
C
C          CALL THE ROUTINES TO TIME THE RUN
C
           LUN=5
           CALL OPEN CPUTIME FILE(LUN)
           CALL START CPU CLOCK
           OPEN(UNIT=1,NAME='N.DAT',TYPE='OLD',
     1     ACCESS='DIRECT',RECORD SIZE=128,MAXREC=640)
           DO 10 M=1,512
           READ(1'M)(A(M,N),N=1,480)
           DO 5 P=481,512
           A(M,P)=0
     5     CONTINUE
    10     CONTINUE
           CLOSE(UNIT=1)
C
C          THE IMAGE IS NOW A 512 X 512 PIXEL ARRAY
C          NEIGHBORHOOD AVERAGING FOLLOWS
C
           DO 68 L=1,5              ;SMOOTH THE IMAGE 5 TIMES
           DO 50 I=2,511
           DO 60 J=2,511
           C(1)=A(I,J-1)
           C(2)=A(I,J+1)
           C(3)=A(I-1,J)
           C(4)=A(I+1,J)
C
C          CONVERT THE DATA FROM BYTE TO INTEGER
C
           DO 33 K=1,4
           IF(C(K).GE.0) THEN
                CC(K)=C(K)
           ELSE
                CC(K)=C(K)+256
           ENDIF
    33     CONTINUE
```

```
C
C          AVERAGE THE 4 NEIGHBORS
C
           ZZ=CC(1)+CC(2)+CC(3)+CC(4)
           Z=ZZ/4
C
C          CONVERT THE AVERAGE TO BYTE
C
           IF(Z.LE.0) THEN
                B(I,J)=Z
           ELSE
                B(I,J)=Z-256
           ENDIF
   60      CONTINUE
   50      CONTINUE
C
C          SWAP THE DATA ARRAYS AND LOOP (5 TIMES)
C
           DO 66 I=2,511
           DO 67 J=2,511
           A(I,J)=B(I,J)
   67      CONTINUE
   66      CONTINUE
   68      CONTINUE
           WRITE(5,35)
   35      FORMAT(' AVERAGING IS COMPLETE')
C
C          THE BORDERS ARE NOW FILLED IN
C
           I=1
           DO 70 J=1,512
           B(I,J)=A(I,J)
           B(J,I)=A(J,I)
           B(I+511,J)=A(I+511,J)
           B(J,I+511)=A(J,I+511)
   70      CONTINUE
C
C          WRITE THE FILE S.DAT TO DISK
C
           OPEN(UNIT=2,NAME='S.DAT',TYPE='NEW',
     1     ACCESS='DIRECT',RECORD SIZE=128,MAXREC=512)
           DO 90 M=1,512
           WRITE(2'M)(B(M,N),N=1,512)
   90      CONTINUE
           CLOSE(UNIT=2)
C
C          WRITE CPU TIME TO THE FILE CPUTIME.TXT
C
           CALL TYPE CPU CLOCK(LUN)
           END
```

```
      SUBROUTINE OPEN CPUTIME FILE(LUN)
                OPEN(UNIT=LUN,NAME='CPUTIME.TXT',STATUS='NEW')
                END

      SUBROUTINE JCPUT(XCPUT)
      C
      C         RETURN CPU TIME AS A FLOATING POINT VALUE
      C
                PARAMETER JPI$ CPUTIM='407'X
                INTEGER*2 BUF(8)
                INTEGER*4 BUF1(4),CPUT
                INTEGER SYS$GETJPI
                EQUIVALENCE(BUF(1),BUF1(1))
                REAL XCOUT
                BUF(1)=4
                BUF(2)=JPI$ CPUTIM
                BUF1(2)=%LOC(CPUT)
                BUF1(3)=0
                BUF1(4)=0
                IRET=SYS$GETJPI(,,,,BUF,,,)
                XCPUT=FLOAT(CPUT)/100.0
                RETURN
                END

      SUBROUTINE START CPU CLOCK
      C
      C         GET THE INITIAL VALUE OF THE CLOCK
      C
                REAL*4 START TIME
                REAL*4 START REAL TIME
                COMMON/CPUT CLOCK/START TIME,START REAL TIME
      C
      C         READ THE ELAPSED TIME AND THE ELAPSED REAL TIME
      C
                CALL JCOUT(START TIME)
      C
      C         GET THE INITIAL REAL TIME
      C
                START REAL TIME=SECNDS(0.0)
                RETURN
                END

      SUBROUTINE TYPE CPU CLOCK(LUN)
      C
      C         RETURN THE ELAPSED TIME FROM THE LAST CALL TO
      C         START CPU CLOCK
      C
                REAL*4 TIME,START TIME
                REAL*4 START REAL TIME
                REAL*4 DELTA
                COMMON/CPU CLOCK/START TIME,START REAL TIME
```

```
C
C          GET THE CPU TIME
C
           CALL JCOUT(TIME)
C
C          COMPUTE THE DELTA TIME
C
           TIME=TIME-START TIME
C
C          GET THE DELTA REAL TIME
C
           DELTA=SECNDS(START REAL TIME)
           XLOAD=TIME/DELTA
C
C          DISPLAY AS A PERCENTAGE
C
           XLOAD=XLOAD/100.0
           WRITE(LUN,10)TIME,DELTA,XLOAD
  10       FORMAT(' CPU TIME USED=',F10.3,
     1     'REAL TIME USED=',F10.3,
     1     'LOAD FACTOR=',F10.4,'%')
           RETURN
           END
```

## APPENDIX B
## MICROCODE DOCUMENTATION FOR EXAMPLE 1

LINE NO.: 0000
OPERATION: CONTINUE

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CI |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#X | ;don't care |
| 11-4 | | ADDRESS | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: FFFF E4F9 FFFE (X=1)

COMMENTS:
This microroutine only demonstrates the use of the sequencer. The remainder of the microinstruction is then set to safe values as shown above. This microword only instructs the sequencer to go to the next sequential instruction in WCS.

LINE NO.: 0001
OPERATION: JUMP TO 0020(H)

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-15 | | CMD | H#9 | ;test AM2904 CT |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#00 | ;branch address MSB |
| 11-4 | | ADDRESS | H#20 | ;branch address LSB |
| 3-0 | AM2910 | INSTR | H#1 | ;CJS |

RESULTING MICROWORD: FFFF E4F9 C201 (X=1)

COMMENTS:
This microinstruction performs an unconditional jump to WCS address 0020(H) as indicated by the branch address bits 4-13 of the pipeline.

LINE NO.: 0020
OPERATION: CONTINUE

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CT |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#X | ;don't care |
| 11-4 | | ADDRESS | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: FFFF E4F9 FFFE (X=1)

COMMENTS:
This microword only instructs the sequencer to go to the next sequential instruction in WCS, i.e., address 0021(H).

LINE NO.: 0021
OPERATION: LOAD THE COUNTER W/4 & CONTINUE

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CI |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#00 | ;counter holds 10 bits |
| 11-4 | | CONSTANT | H#04 | ;load it with 4 |
| 3-0 | AM2910 | INSTR | H#C | ;ldct w/4 & continue |

RESULTING MICROWORD: FFFF E4F9 C04C (X=1)

COMMENTS:
This microword instructs the sequencer to load the counter with the value 4 and to go to the next sequential instruction in WCS, i.e., address 0022(H).

LINE NO.: 0022
OPERATION: LOOP UNTIL COUNTER = 0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CI |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#00 | ;address MSB |
| 11-4 | | ADDRESS | H#22 | ;address LSB |
| 3-0 | AM2910 | INSTR | H#9 | ;loop until ctr=0 |

RESULTING MICROWORD: FFFF E4F9 C229 (X=1)

COMMENTS:
This microword instructs the sequencer to loop to address 0022(H) decrimenting the counter each time and continuing to the next sequential instruction once the counter equals zero. There will be a total of 5 loops.

LINE NO.: 0023
OPERATION: UNCONDITIONAL SUBR CALL TO 0200(H)

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CI |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#01 | ;address MSB |
| 11-4 | | ADDRESS | H#00 | ;address LSB |
| 3-0 | AM2910 | INSTR | H#3 | ;CJP |

RESULTING MICROWORD: FFFF E4F9 E003 (X=1)

COMMENTS:
This microword instructs the sequencer to perform an unconditional subroutine call to WCS address 0200(H). On the subsequent return, the sequencer will jump to WCS address 0024(H) which will be stored on the stack with the call and popped off the stack with the return.

LINE NO.: 0024
OPERATION: JUMP TO 0000(H)

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CT |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#00 | ;branch address MSB |
| 11-4 | | ADDRESS | H#00 | ;branch address LSB |
| 3-0 | AM2910 | INSTR | H#1 | ;CJS |

RESULTING MICROWORD: FFFF E4F9 C001 (X=1)

COMMENTS:
This microinstruction performs an unconditional jump to WCS address 0000(H) as indicated by the branch address bits 4-13 of the pipeline.

LINE NO.: 0200
OPERATION: RETURN

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#X | ;don't care |
| 29-24 | | STAT/TST | Q#44 | ;test macro zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;command |
| 19-16 | | CMD | H#9 | ;test AM2904 CT |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | ADDRESS | B#XX | ;don't care |
| 11-4 | | ADDRESS | H#XX | ;don't care |
| 3-0 | AM2910 | INSTR | H#3 | ;CRTN |

RESULTING MICROWORD: FFFF E4F9 FFFA (X=1)

COMMENTS:
This microword instructs the sequencer to pop the top address off the stack and return to the line following the command that called the subroutine.

# APPENDIX C
# DOCUMENTATION FOR THE 5 X 5 MICROROUTINE

LINE NO.: 0100
OPERATION: PUSH ADD. ON STACK, LD CTR W/02

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#1 | ;disable Am29203 |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#00 | ;upper 2 bits of counter |
| 11-8 | REGSEL | RA | H#0 | ;counter data |
| 7-4 | | RB | H#2 | ;load counter with 2 |
| 3-0 | AM2910 | INSTR | H#4 | ;push & ld ctr, continue |

RESULTING MICROWORD: FFFF 3FFF C024 (X=1)

COMMENTS:
This instruction only involves the use of the sequencer, hence all the don't cares through the documentation. This instruction sets the address 0100(H) on the stack. This address is returned to on the outer loop, loop1. The counter is also loaded in this microcycle so that the inner loop will execute three times.

LINE NO.: 0101
OPERATION: MEMORY -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint. |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 084F 3FD3 F14E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is a noop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R4. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R4. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the neighbor from memory and stores the value into R4. R4 is to be the accumulator for the four reads from memory.

LINE NO.: 0102
OPERATION: R0 + R1 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#0 | ;RA=R0 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F01E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R0 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the register that holds the address of the neighbor to be read. R1 is initialized to 0001(H) for the first read and is now incrimented by the value of R0, 4, to the next address to be read.

LINE NO.: 0103
OPERATION: MEMORY -> R3

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#3 | ;data destination R3 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 084F 3FD3 F13E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R3. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R3. The function (bits 32-35) is a noop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R3. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R3. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the neighbor from memory and stores the value into R3.

LINE NO.: 0104
OPERATION: R3 + R4 -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-15 | | CMC | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#3 | ;RA=R3 |
| 7-4 | | RB | H#4 | ;RB=R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F34E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R3 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R4. The Am2910 is instructed to continue to the next sequential instruction. R3 holds the value of the neighbor read from RAM and is added to R4 which is the accumulator for the addition of the four neighbors.

LINE NO.: 0105
OPERATION: R1 + R2 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#2 | ;RA=R2 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F21E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R2 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. The address of the next neighbor to be read is incrimented by 2 to read the neighbor to the right of the subject pixel.

LINE NO.: 0106
OPERATION: MEMORY -> R3

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to Y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | Y | ;not used |
| 13-12 | | CONSTANT | B#YY | ;not used |
| 11-8 | REGSEL | PA | H#1 | ;memory address in R1 |
| 7-4 | | PB | H#3 | ;data destination R3 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICRO-OPD: 094F 3FD3 F13E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set PA=R1 and PB=R3. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is PB=R3. The function (bits 32-35) is a nood since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R3. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R3. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the neighbor from memory and stores the value into R3.

LINE NO.: 0107
OPERATION: R3 + R4 -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#3 | ;RA=R3 |
| 7-4 | | RB | H#4 | ;RB=R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F34E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R3 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R4. The Am2910 is instructed to continue to the next sequential instruction. R3 holds the value of the neighbor read from RAM and is added to R4 which is the accumulator for the addition of the four neighbors.

LINE NO.: 0108
OPERATION: R0 + R1 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#0 | ;RA=R0 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FEF F01E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R0 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the register that holds the address of the neighbor to be read. R1 is initialized to 0001(H) for the first read and is now incrimented by the value of R0, 4, to the next address to be read.

LINE NO.: 0109
OPERATION: MEMORY -> R3

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#3 | ;data destination R3 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 084F 3FD3 F13E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R3. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R3. The function (bits 32-35) is a ncop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R3. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R3. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the neighbor from memory and stores the value into R3.

LINE NO.: 010A
OPERATION: R3 + R4 -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#3 | ;RA=R3 |
| 7-4 | | RB | H#4 | ;RB=R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF F34E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R3 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R4. The Am2910 is instructed to continue to the next sequential instruction. R3 holds the value of the neighbor read from RAM and is added to R4 which is the accumulator for the addition of the four neighbors.

LINE NO.: 010B
OPERATION: LOGICAL SHIFT RIGHT OF R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;regs. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;registers |
| 39-36 | | DEST | H#1 | ;regs., log. downshift |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#10 | ;shift |
| 19-16 | | CMD | H#C | ;shift left, bring in 0 |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;not used |
| 7-4 | | RB | H#4 | ;shift R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0014 3FE0 FF4E (X=1)

COMMENTS:
This instruction enables the ALU only to allow the passing of the data to be shifted, R4. R4 is sent onto the Y bus and is passed through the AM2904 which shifts the bits to the right and fills with a zero. The shifted value is then put back into R4. This accomplishes a divide by two.

LINE NO.: 010C
OPERATION: LOGICAL SHIFT RIGHT OF R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;regs. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;registers |
| 39-36 | | DEST | H#1 | ;regs., log. downshift |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#10 | ;shift |
| 19-16 | | CMD | H#0 | ;shift left, bring in 0 |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;not used |
| 7-4 | | RB | H#4 | ;shift R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0014 3FE0 FF4E (X=1)

COMMENTS:
This instruction enables the ALU only to allow the passing of the data to be shifted, R4. R4 is sent onto the Y bus and is passed through the AM2904 which shifts the bits to the right and fills with a zero. The shifted value is then put back into R4. This accomplishes a divide by two.

LINE NO.: 0100
OPERATION: R4 -> MEMORY

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#C | ;result to y bus only |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory write |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#7 | ;memory address in R7 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 00C4 3FD4 F74E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R7 and RB=R4. The function (bits 32-35) is a pass through since the ALU is only to put the data on the Y bus. The command field is enabled to write to memory. The address to be written to is held in R7 and the contents to be written are in R4. The Am2910 is instructed to continue to the next sequential instruction. This operation writes the average to the smoothed array in memory.

LINE NO.: 010E
OPERATION: R1 - R6 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;regs. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;registers |
| 39-36 | | DEST | H#4 | ;result to Y bus & B reg |
| 35-32 | | FUNCT | H#1 | ;subtract |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#6 | ;RA=R6 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0041 3FDF F61E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R6 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is a subtract of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the register that holds the address of the neighbor to be read. R1 is at the end of one set of four reads and must be reinitialized back in the array so that four new reads from memory can again be executed. It is decremented by 8.

LINE NO.: 010F
OPERATION: INCRIMENT R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;increment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the address that the next average will be stored into. This is the address of the smoothed array.

LINE NO.: 0110
OPERATION: DEC. COUNTER, JUMP TO LOOP1 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;no command |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#8 | ;dec. counter, cont. >0 |

RESULTING MICROWORD: FFFF 3FFF FFF8 (X=1)

COMMENTS:
At this point, the first average has been written to the smoothed array. This is to be done a total of three times per row. The counter is decremented and tested for zero. If not yet zero, the sequencer loops back to address 0000(H). If it is zero, the sequencer continues to the next sequential instruction.

LINE NO.: 0111
OPERATION: P1 + P2 -> P1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | 3#0 | reg. used by pipeline |
| 44 | AM29203 | IEN | B#0 | enable AM29203 |
| 43 | | OEY | B#0 | connect Y bus |
| 42-40 | | SOURCE | 3#0 | sources are regs |
| 39-36 | | DEST | H#4 | result to A & B reg |
| 35-32 | | FUNCT | H#3 | add |
| 31-30 | AM2904 | CARRY | B#00 | no carry in |
| 29-24 | | STAT IST | 3#1 | don't care |
| 23 | | CEU | B#1 | don't latch micro stat |
| 22 | | CEM | B#1 | don't latch macro stat |
| 21-20 | | CNDSHFT | B#01 | command enable |
| 19-16 | | CND | H#F | mode |
| 15 | | BKPT | B#1 | don't set breakpoint |
| 14 | | SPARE | 1 | not used |
| 13-12 | | CONSTANT | B#11 | not used |
| 11-8 | REGSEL | RA | H#2 | RA=R2 |
| 7-4 | | RB | H#1 | RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | continue |

RESULTING MICROWORD: 0043 3E0E F21E 4E1

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R2 and RB=R1. The ALU source bits 40-42 are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The AM2910 is instructed to continue to the next sequential instruction. The address of the next neighbor to be read is incremented by 2 to get by the border values which will be dealt with later. At this point, a row of three averages have been written to the smoothed array.

LINE NO.: 0112
OPERATION: R7 + R2 -> R7

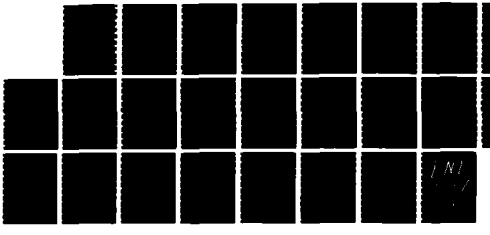
| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#2 | ;RA=R2 |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

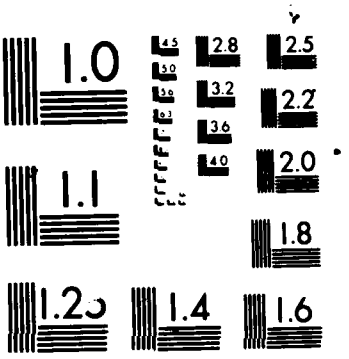
RESULTING MICROWORD: 0043 3FDF F27E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to [illegible] pipeline. Therefore, bits 4-11 set RA=R2 and [illegible] ALU source (bits 40-42) are these registers. The destination (bits 36-39) is RB=R7. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. The [illegible] next average is to written is incremented. This moves that pointer by the border [illegible] dealt with later.

AD-A176 853 THE APPLICATION OF BIT SLICE DESIGN TO DIGITAL IMAGE PROCESSING(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M B STEWART SEP 86 2/2

UNCLASSIFIED F/G 9/2 NL





LINE NO.: 0113
OPERATION: R8 - 1 -> R8, LATCH MICROSTAT REGISTER

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#3 | ;decriment by 1 |
| 35-32 | | FUNCT | H#0 | ;special function |
| 31-30 | AM2904 | CARRY | B#01 | ;decriment by 1 |
| 29-24 | | STAT/TST | Q#XX | ;latch ALU output |
| 23 | | CEU | B#1 | ;latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#1 | ;RB=R8 |
| 3-0 | AM2910 | INSTR | H#8 | ;continue |

RESULTING MICROWORD: 0030 507F FF8E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R8. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R8. The function (bits 32-35) is a decriment of the source register with the result being sent to RB=R8. The Am2910 is instructed to continue to the next sequential instruction. The microstatus register is also latched. It will be tested on the next microcycle for zero. This is the outer loop test.

LINE NO.: 0114
OPERATION: DEC. COUNTER, JUMP TO LOOP1 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#24 | ;test micro-zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;enable command |
| 19-16 | | CMD | H#9 | ;test AM2904 CI |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#01 | ;MSB of loop address |
| 11-8 | REGSEL | RA | H#0 | ;loop address |
| 7-4 | | RB | H#0 | ;loop address |
| 3-0 | AM2910 | INSTR | H#3 | ;CJP to loop1 / test neg |

RESULTING MICROWORD: FFFF D4D9 D003 (X=1)

COMMENTS:
At this point, the three averages haves been written to the smoothed array. This is to be done a total of three times per row. Once the three rows have been completed, the sequence is to continue. This instruction tests the results of the decriment of R8, the outer loop counter. If the result was zero, the sequence continues. If not, the sequencer loops back to loop1

LINE NO.: 0115
OPERATION: R7 - R5 -> R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;regs. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;registers |
| 39-36 | | DEST | H#4 | ;result to Y bus & B reg |
| 35-32 | | FUNCT | H#1 | ;subtract |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#5 | ;RA=R5 |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0041 3FDF F57E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R5 and RB=R7. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is a subtract of the source registers with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the register that holds the address where the averages are to be stored. At this point, all the averages have been stored and the border must now be written. This register must now be decremented back to the starting address of the array, i.e., decremented by 14(H).

LINE NO.: 0116
OPERATION: R1 - R9 -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;regs. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;registers |
| 39-36 | | DEST | H#4 | ;result to Y bus & B reg |
| 35-32 | | FUNCT | H#1 | ;subtract |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#9 | ;RA=R9 |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0041 3FDF F91E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R9 and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is a subtract of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the register that holds the address where the original array elements exist. The averaging is complete and now it is time to write the border values. The starting address of the original array is held in R1 and is found by subtracting OF(H) from the register's present contents.

LINE NO.: 0117
OPERATION: PUSH ADD. ON STACK, LD CTR W/05

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#1 | ;disable Am29203 |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#00 | ;upper 2 bits of counter |
| 11-8 | REGSEL | RA | H#0 | ;counter data |
| 7-4 | | RB | H#5 | ;load counter with 5 |
| 3-0 | AM2910 | INSTR | H#4 | ;push & ld ctr, continue |

RESULTING MICROWORD: FFFF 3FFF C054 (X=1)

COMMENTS:
This instruction only involves the use of the sequencer, hence all the don't cares through the documentation. This instruction pushes the address 0117(H) on the stack. This address is returned to as loop2. The counter is also loaded in this microcycle so that the loop will execute six times. This loop reads and write the first six elements of each array, the border values.

LINE NO.: 0118
OPERATION: MEMORY -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 084F 3FD3 F14E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is a noop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R4. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R4. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the border value from memory and stores the value into R4. R4 is to be then written to the smoothed array.

LINE NO.: 0119
OPERATION: R4 -> MEMORY

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#C | ;result to y bus only |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory write |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#7 | ;memory address in R7 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 00C4 3FD4 F74E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R7 and RB=R4. The function (bits 32-35) is a pass through since the ALU is only to put the data on the Y bus. The command field is enabled to write to memory. The address to be written to is held in R7 and the contents to be written are in R4. The Am2910 is instructed to continue to the next sequential instruction. This operation writes the border value to the smoothed array in memory.

LINE NO.: 011A
OPERATION: INCRIMENT R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF1E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R1 is the address that the next border value will be read from.

LINE NO.: 011B
OPERATION: INCRIMENT R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the address that the next border value will be stored into. This is the address of the smoothed array.

LINE NO.: 011C
OPERATION: DEC. COUNTER, JUMP TO LOOP2 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;no command |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#8 | ;dec. counter, cont. >0 |

RESULTING MICROWORD: FFFF 3FFF FFF8 (X=1)

COMMENTS:
At this point, the first border value has been written to the smoothed array. This is to be done a total of six times. The counter is decremented and tested for zero. If not yet zero, the sequencer loops back to address 0117(H). If it is zero, the sequencer continues to the next sequential instruction.

LINE NO.: 011D
OPERATION: LOAD IR W/ADDRESS OF R8

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#1 | ;disable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#5 | ;need DB as source |
| 39-36 | | DEST | H#C | ;don't load the register |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;enable command |
| 19-16 | | CMD | H#2 | ;constant to IR |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;don't care |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#8 | ;address of R8 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: FSC4 3FD2 FF8E (X=1)

COMMENTS:
This instruction loads the address of R8, 0008, into the IR. This is necessary so that this outer loop counter can be reset to count the outer loop of another nested loop. This method of reseting the contents of a register will be shown in the next microinstruction.

LINE NO.: 011E
OPERATION: LOAD R8 W/02

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#7 | ;address from IR |
| 44 | AM29203 | IEN | B#0 | ;enable write |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#5 | ;need DB input |
| 39-36 | | DEST | H#4 | ;register |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;enable command |
| 19-16 | | CMD | H#5 | ;constant to DS-R8 |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#0 | ;loading data |
| 7-4 | | RB | H#2 | ;load register with 2 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: E544 3FD5 F02E (X=1)

COMMENTS:
Having stored the address of R8 into the IR, the value 02 is then sent to that register for loading. This number is then used as a loop counter to be decremented with each passing.

LINE NO.: 011F
OPERATION: R1 + RA -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#A | ;RA=RA |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF FA1E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=RA and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the register that holds the address of the neighbor to be read. R1 is incrimented by three to pass over the averaged values and load the border values for writing over to the smoothed array.

LINE NO.: 0120
OPERATION: R7 + RA -> R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#A | ;RA=RA |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF FA7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=RA and RB=R7. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the register that holds the address where the next border value is to be written. It must be incrimented by three to pass over the averaged values already written to the smoothed array.

LINE NO.: 0121
OPERATION: PUSH ADD. ON STACK, LD CTR W/01

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#1 | ;disable Am29203 |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#00 | ;upper 2 bits of counter |
| 11-8 | REGSEL | RA | H#0 | ;counter data |
| 7-4 | | RB | H#1 | ;load counter with 1 |
| 3-0 | AM2910 | INSTR | H#4 | ;push & ld ctr, continue |

RESULTING MICROWORD: FFFF 3FFF C014 (X=1)

COMMENTS:
This instruction only involves the use of the sequencer, hence all the don't cares through the documentation. This instruction sets the address 0121(H) on the stack. This address is returned to on the loop4. The counter is also loaded in this microcycle so that the loop will execute two times.

LINE NO.: 0122
OPERATION: MEMORY -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 084F 3FD3 F14E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is a noop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R4. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R4. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the border value from memory and stores the value into R4.

LINE NO.: 0123
OPERATION: R4 -> MEMORY

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#C | ;result to y bus only |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory write |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#7 | ;memory address in R7 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 00C4 3FD4 F74E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R7 and RB=R4. The function (bits 32-35) is a pass through since the ALU is only to put the data on the Y bus. The command field is enabled to write to memory. The address to be written to is held in R7 and the contents to be written are in R4. The Am2910 is instructed to continue to the next sequential instruction. This operation writes the border value to the smoothed array in memory.

LINE NO.: 0124
OPERATION: INCRIMENT R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/IST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF1E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. R1 is the address that the next border value will be read from.

LINE NO.: 0125
OPERATION: INCRIMENT R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R7. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the address that the next border value will be stored into. This is the address of the smoothed array.

LINE NO.: 0126
OPERATION: DEC. COUNTER, JUMP TO LOOP1 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TSI | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;no command |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#8 | ;dec. counter, cont. >0 |

RESULTING MICROWORD: FFFF 3FFF FFF8 (X=1)

COMMENTS:
The counter is decrimented and tested for zero. If not yet zero, the sequencer loops back to address 0121(H). If it is zero, the sequencer continues to the next sequential instruction.

LINE NO.: 0127
OPERATION: R8 - 1 -> R8, LATCH MICROSTAT REGISTER

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#3 | ;decriment by 1 |
| 35-32 | | FUNCT | H#0 | ;special function |
| 31-30 | AM2904 | CARRY | B#01 | ;decriment by 1 |
| 29-24 | | STAT/TST | Q#XX | ;latch ALU output |
| 23 | | CEU | B#1 | ;latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#1 | ;RB=R8 |
| 3-0 | AM2910 | INSTR | H#8 | ;continue |

RESULTING MICROWORD: 0030 507F FF8E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R8. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R8. The function (bits 32-35) is a decriment of the source register with the result being sent to RB=R8. The Am2910 is instructed to continue to the next sequential instruction. The microstatus register is also latched. It will be tested on the next microcycle for zero. This is the outer loop test.

LINE NO.: 0128
OPERATION: DEC. COUNTER, JUMP TO LOOP3 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#24 | ;test micro-zero |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;enable command |
| 19-16 | | CMD | H#9 | ;test AM2904 CT |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#01 | ;MSB of loop address |
| 11-8 | REGSEL | RA | H#1 | ;loop address |
| 7-4 | | RB | H#F | ;loop address |
| 3-0 | AM2910 | INSTR | H#3 | ;CJP to loop3 / test neg |

RESULTING MICROWORD: FFFF D4D9 D1F3 (X=1)

COMMENTS:
This instruction tests the results of the decriment of R9, the outer loop counter. If the result was zero, the sequence continues. If not, the sequencer loops back to loop3.

LINE NO.: 0129
OPERATION: R1 + RA -> R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#A | ;RA=RA |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF FA1E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=RA and RB=R1. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R1. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R1. The Am2910 is instructed to continue to the next sequential instruction. The address of the next neighbor to be read is incrimented by 3 to get by the averaged values.

LINE NO.: 012A
OPERATION: R7 + RA -> R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#3 | ;add |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#F | ;noop |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#A | ;RA=RA |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0043 3FDF FA7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=RA and RB=R7. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an add of the source registers with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. The address where the next border value is to be stored is incrimented by 3 to pass over the averaged values.

LINE NO.: 012B
OPERATION: PUSH ADD. ON STACK, LD CTR W/OS

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#1 | ;disable Am29203 |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#11 | ;no command or shift |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#00 | ;upper 2 bits of counter |
| 11-8 | REGSEL | RA | H#0 | ;counter data |
| 7-4 | | RB | H#5 | ;load counter with 5 |
| 3-0 | AM2910 | INSTR | H#4 | ;push & ld ctr, continue |

RESULTING MICROWORD: FFFF 3FFF C054 (X=1)

COMMENTS:
This instruction only involves the use of the sequencer, hence all the don't cares through the documentation. This instruction pushes the address 012B(H) on the stack. This address is returned to as loop5. The counter is also loaded in this microcycle so that the loop will execute six times. This loop reads and write the last six elements of each array, the border values.

LINE NO.: 012C
OPERATION: MEMORY -> R4

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#1 | ;disconnect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/IST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory read |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#1 | ;memory address in R1 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: C84F 3FD3 F14E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R1 and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R4. The function (bits 32-35) is a noop since the ALU is not connected to the Y bus. The ALU is enabled only to allow the loading of the data to R4. The command field is enabled to read from memory. The address to be read is held in R1 and the contents of that address are to be sent to R4. The Am2910 is instructed to continue to the next sequential instruction. This operation reads the border value from memory and stores the value into R4. R4 is to be then written to the smoothed array.

LINE NO.: 012D
OPERATION: R4 -> MEMORY

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#C | ;result to y bus only |
| 35-32 | | FUNCT | H#4 | ;pass through |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#01 | ;command enable |
| 19-16 | | CMD | H#3 | ;memory write |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#7 | ;memory address in R7 |
| 7-4 | | RB | H#4 | ;data destination R4 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 00C4 3FD4 F74E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=R7 and RB=R4. The function (bits 32-35) is a pass through since the ALU is only to put the data on the Y bus. The command field is enabled to write to memory. The address to be written to is held in R7 and the contents to be written are in R4. The Am2910 is instructed to continue to the next sequential instruction. This operation writes the border value to the smoothed array in memory.

LINE NO.: 012E
OPERATION: INCRIMENT R1

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#1 | ;RB=R1 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7FFF FF1E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R1 is the address that the next border value will be read from.

LINE NO.: 012F
OPERATION: INCRIMENT R7

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#0 | ;reg. spec. by pipeline |
| 44 | AM29203 | IEN | B#0 | ;enable Am29203 |
| 43 | | OEY | B#0 | ;connect Y bus |
| 42-40 | | SOURCE | Q#0 | ;sources are regs. |
| 39-36 | | DEST | H#4 | ;result to y & B-reg |
| 35-32 | | FUNCT | H#4 | ;incriment |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;don't care |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#7 | ;RB=R7 |
| 3-0 | AM2910 | INSTR | H#E | ;continue |

RESULTING MICROWORD: 0044 7EFF FF7E (X=1)

COMMENTS:
Bits 45-47 declare the source registers to be in the pipeline. Therefore, bits 4-11 set RA=XX and RB=R4. The ALU source (bits 40-42) are these registers and the destination (bits 36-39) is RB=R7. The function (bits 32-35) is an incrment of the source register with the result being sent to RB=R7. The Am2910 is instructed to continue to the next sequential instruction. R7 is the address that the next border value will be stored into. This is the address of the smoothed array.

LINE NO.: 0130
OPERATION: DEC. COUNTER, JUMP TO LOOP5 IF >0

| BITS | DEVICE | FIELD | VALUE | EXPLANATION |
|---|---|---|---|---|
| 47-45 | | REGSRC | Q#X | ;don't care |
| 44 | AM29203 | IEN | B#X | ;don't care |
| 43 | | OEY | B#X | ;don't care |
| 42-40 | | SOURCE | Q#X | ;don't care |
| 39-36 | | DEST | H#X | ;don't care |
| 35-32 | | FUNCT | H#X | ;don't care |
| 31-30 | AM2904 | CARRY | B#00 | ;no carry in |
| 29-24 | | STAT/TST | Q#XX | ;don't care |
| 23 | | CEU | B#1 | ;don't latch micro stat |
| 22 | | CEM | B#1 | ;don't latch macro stat |
| 21-20 | | CMDSHFT | B#XX | ;no command |
| 19-16 | | CMD | H#X | ;don't care |
| 15 | | BKPT | B#1 | ;don't set breakpoint |
| 14 | | SPARE | X | ;not used |
| 13-12 | | CONSTANT | B#XX | ;not used |
| 11-8 | REGSEL | RA | H#X | ;don't care |
| 7-4 | | RB | H#X | ;don't care |
| 3-0 | AM2910 | INSTR | H#8 | ;dec. counter, cont. >0 |

RESULTING MICROWORD: FFFF 3FFF FFF8 (X=1)

COMMENTS:
At this point, the a border value has been written to the smoothed array. This is to be done a total of six times. The counter is decrimented and tested for zero. If not yet zero, the sequencer loops back to address 012B(H). If it is zero, the sequencer continues to the next sequential instruction which is a breakpoint, i.e., the routine is halted.

## LIST OF REF CES


1. Wilkinson, D. A., "Mappir he Earth in Bits and Bytes", Defense/85, p. 22, May 1985.

2. Defense Mapping Agency Hydrographic/Topographic Center Solicitation Number DMA800-86-R-0126, Information to Offerors or Qouters, January 1986.

3. Wilson, A. C., "Bring on the 90s", The S. Klein Computer Graphics Review, p. 21, Inaugural Issue.

4. Wilson, A. C., "Bring on the 90s", p. 14.

5. Siwolop, S., "Transputer Chips: Linchpins of a Mighty Supercomputer", Business Week, p. 47, 21 April, 1986.

6. Am29203 Evaluation Board User's Guide, p. 4-2, Advanced Micro Devices, Inc., 1986.

BIBLIOGRAPHY


1. White, D. E., Bit-Slice Design: Controllers and ALUs, Garland STPM Press, 1981.

2. Bipolar Microprocessor and Interface Data Book, Advanced Micro Devices Inc., 1985.

3. Gonzalez, R. C., Digital Image Processing, Addison-Wesley Publishing Company, 1977.

4. Pavlidis, T., Algorithms for Graphics and Image Processing, Computer Science Press, 1982.

INITIAL DISTRIBUTION LIST

| | | No. of Copies |
|---|---|---|
| 1. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5002 | 2 |
| 2. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 3. | Department Chairman, Code 62<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 4. | Professor Chin-Hwa Lee, Code 62Le<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 4 |
| 5. | Professor Mitchell Cotton, Code 62Cc<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 6. | Commandant (G-PTE)<br>U. S. Coast Guard<br>2100 Second St. S. W.<br>Washington, D. C. 20593 | 2 |
| 7. | Commanding Officer<br>U. S. Coast Guard Electronics Engineering Center<br>P. O. Box 60<br>Wildwood, N. J. 08260-0060<br>Attn: LTJG M. B. Stewart | 2 |

END

3-87

DTIC